A NEURAL NETWORK CONSTRUCTION METHOD FOR SURROGATE MODELING OF PHYSICS-BASED ANALYSIS

A Thesis
Presented to
The Academic Faculty

by

Woong Je Sung

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2012

A NEURAL NETWORK CONSTRUCTION METHOD FOR SURROGATE MODELING OF PHYSICS-BASED ANALYSIS

Approved by:

Professor Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Assistant Professor Vitali Volovoi
School of Aerospace Engineering
Georgia Institute of Technology

Assistant Professor Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Associate Professor Sung Ha Kang
School of Mathematics
Georgia Institute of Technology

Associate Professor Mark Costello
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: April 2012

To my parents, Baek Woon Sung and Soon Ja Kim

who always believe in their son.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Dimitri Mavris, for all his guidance and support throughout this journey. I thank staff and colleagues in the Aerospace Systems Design Laboratory for their direct and indirect encouragement throughout my dissertation. I would like to express my gratitude to two special teachers in my life, the late Mrs. Kyung Soon Suh and Dr. Sung Joon Yoon. They were always alive in my heart during my studies. I am always grateful to my parents, Baek Woon Sung and Soon Ja Kim, for their relentless devotion and support for my education. My sincere thanks also goes to my loving wife, Hye Shin, for her persistent encouragement and support for me. I would like to thank my son and daughter, Siho and Yuri, for always being a cheerful source of inspiration for my studies. Last but not least, I would like to thank my sister, Jee Yun, for always being a strong supporter, whatever her brother studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Modeling and Simulation (M&S) tasks in aeronautics are increasingly demanding high-fidelity, physics-based analysis tools to be integrated as component modules to fulfill ever-increasing technological challenges. These challenges are being derived from the need for the intelligent and adaptive aeronautical systems that greatly improve the performance and robustness of aircraft and the air transportation system as a whole. The current level of the multidisciplinary design optimization (MDO) methodologies has been matured for conventional system designs and became an important part of design process but for problems with topological multiplicities and for problems involving large number of design variables, they are still underdeveloped. Particularly, the integration of high-fidelity, physics-based analysis modules such as computational fluid dynamics (CFD) codes into the MDO frameworks is only practical within the restricted design space because of the prohibitive requirement for the computational resources as the design freedom increases.

The Artificial Neural Network (ANN) is expanding its application area to the fluid mechanics field due to its promising mapping and prediction capability for the highly complex, nonlinear, unsteady flow phenomena using relatively small amount of training data. This generalization capability of the ANN is based on its adaptive learning ability that results from the network built out of the many, extremely simple computational units. A growing number of researchers are exploring the possibility of the previously computationally prohibitive merge between stochastic design optimization techniques and high-fidelity, physics-based analysis modules. The key enabler in this direction of research is the efficient surrogate modeling via the ANN techniques.

However, to maximize the advantages of the ANN surrogate modeling, several fundamental subjects still need to be investigated further. While there are many effective methods for optimization of the network weight parameters for the pre-defined network structure, there are only very limited choices on the methods to determine the proper network size and its internal connection architecture. This is an important gap considering that the ANN's generalization capability is strongly dependent on its size and connection structure. Therefore, the lack of the proper means to select the architecture of the ANN greatly hinders the systematic and principled assessment for the generalization efficiency. Hence, its further improvement is no more than being guided by the judicious selection from the handful of stereotyped architectures.

In this thesis the existing methodologies related to the developmental methods of neural networks have been surveyed and their approaches to network sizing and structuring are carefully observed. This literature review covers the constructive methods, the pruning methods, and the evolutionary methods and questions about the basic assumption intrinsic to the conventional neural network learning paradigm, which is primarily devoted to optimization of connection weights (or synaptic strengths) for the pre-determined connection structure of the network. The main research hypothesis governing this thesis is that, without breaking a prevailing dichotomy between weights and connectivity of the network during learning phase, the efficient design of a task-specific neural network is hard to achieve because, as long as connectivity and weights are searched by separate means, a structural optimization of the neural network requires either repetitive re-training procedures or computationally expensive topological meta-search cycles. The establishment and attempted proof of this hypothesis have also partly been inspired by elementary facts in the field of neuroscience which tell us that almost every functioning biological neural network adjusts its synaptic connectivity as well as synaptic strengths, and both learning activities are vital to the very survival as a living organism. In other words, the current learning

paradigm pursues learning of only one aspect of network plasticity leaving the other to the network designer's responsibility representing a fundamental difference from the biological case.

The main contribution of this thesis is designing and testing a novel learning mechanism which efficiently learns not only weight parameters but also connection structure from a given training data set, and positioning this learning mechanism within the surrogate modeling practice. In this work, a simple and straightforward extension to the conventional error Back-Propagation (BP) algorithm has been formulated to enable a simultaneous learning for both connectivity and weights of the Generalized Multilayer Perceptron (GMLP) in supervised learning tasks. A particular objective is to achieve a task-specific network having reasonable generalization performance with a minimal training time. The dichotomy between architectural design and weight optimization is reconciled by a mechanism establishing a new connection for a neuron pair which has potentially higher error-gradient than one of the existing connections. Interpreting an instance of the absence of connection as a zero-weight connection, the potential contribution to training error reduction of any present or absent connection can readily be evaluated using the BP algorithm. Instead of being broken, the connections that contribute less remain *frozen* with constant weight values optimized to that point but they are excluded from further weight optimization until reselected. In this way, a selective weight optimization is executed only for the dynamically maintained pool of high gradient connections. By searching the rapidly changing weights and concentrating optimization resources on them, the learning process is accelerated without either a significant increase in computational cost or a need for re-training. This results in a more task-adapted network connection structure. Combined with another important criterion for the division of a neuron which adds a new computational unit to a network, a highly fitted network can be grown out of the minimal random structure. This particular learning strategy can belong

to a more broad class of the variable connectivity learning scheme and the devised algorithm has been named Optimal Brain Growth (OBG).

The OBG algorithm has been tested on two canonical problems; a regression analysis using the Complicated Interaction Regression Function and a classification of the Two-Spiral Problem. A comparative study with conventional Multilayer Perceptrons (MLPs) consisting of single- and double-hidden layers shows that OBG is less sensitive to random initial conditions and generalizes better with only a minimal increase in computational time. This partially proves that a variable connectivity learning scheme has great potential to enhance computational efficiency and reduce efforts to select proper network architecture.

To investigate the applicability of the OBG to more practical surrogate modeling tasks, the geometry-to-pressure mapping of a particular class of airfoils in the transonic flow regime has been sought using both the conventional MLP networks with pre-defined architecture and the OBG-developed networks started from the same initial MLP networks. Considering wide variety in airfoil geometry and diversity of flow conditions distributed over a range of flow Mach numbers and angles of attack, the new method shows a great potential to capture fundamentally nonlinear flow phenomena especially related to the occurrence of shock waves on airfoil surfaces in transonic flow regime.

These results partially prove the advantage of variable connectivity learning scheme that contrasts to the conventional learning paradigm that is fundamentally synonymous to a weight optimization. By adjusting connectivity and weights seamlessly, within a given computational resource, the increased generalization capability and robustness to initial random factors have been obtained. Therefore, not only can the OBG instantly replace some of the conventional MLP's uses but the basic idea behind the OBG algorithm might also enable a more principled method for sizing and structuring of the general artificial neural networks.

"If the brain were so simple we could understand it, we would be so simple we couldn't." - Lyall Watson

CHAPTER I

INTRODUCTION

An organized investigation for the future technological challenges in civil aeronautics [81] signifies the anticipated evolution of contemporary aeronautical systems into significantly more complex ones. In essence, these challenges are towards the intelligent and adaptive aeronautical systems that enable great improvement of performance and robustness of aircraft and air transportation system as a whole. To meet this future need, the multidisciplinary design optimization (MDO) methodologies need to be improved to integrate high-fidelity analysis modules by efficient design methods and to accommodate uncertainty, multiple objectives, and large-scale systems. Another prediction for these future intelligent and adaptive aeronautical systems by Faller and Schreck is more vivid [26]; *"Potentially, 1,000s of sensors might be utilized to monitor the operational system, and control system commands would drive not only the mechanical actuators but reference models of the plant. The reference models, in turn, would provide a prediction of both the anticipated vehicle dynamics and the expected sensor measurements. Discrepancies between the predicted and measured values could then be utilized both for fault diagnostics and for model reference adaptive controls. Such systems might be expected to expand the safe maneuvering envelope through the use of adaptive control systems, while simultaneously reducing maintenance costs and fault-related accidents through the use of fault diagnostic systems."* In the near future, these intelligent and adaptive aeronautical systems might require aerodynamic shapes to be optimized for the time-dependent operational events responding to the massive sensory inputs for continuously varying flight conditions. Such capability is surely beyond the current paradigm of aerodynamic shape optimization even without

the consideration for another level of multidisciplinary nature interacting with other major disciplines. In the area of computational modeling and simulation (M&S), the quantitative enhancement in computational speed and power and the forecasted reduction of computational time and cost in the near future may partially serve for the development of these future systems. However, a new way of M&S methodology is needed to overcome the current and future limitation imposed by conventional deterministic computation paradigm.

1.1 Surrogate Modeling

The current practice of developing new aircraft is known to depend upon the data collected from approximately 2.5 million aerodynamic experiments and the importance of mathematical modeling is paramount due to its potential for enabling right and efficient decisions on new aircraft designs [6]. For the last several decades, the computational M&S in aeronautics has evolved into MDO methodologies integrating physics-based analysis modules. The term 'physics-based' refers to the prediction of physical phenomena by solving the fundamental equations based on first-principle physical models which contrasts to empiricism at the analysis level [81]. MDO methodologies coupling multiple disciplines in a restricted design space have reached a level of maturity and fidelity, which make them useful as a part of the design process. For example, Computational Fluid Dynamics (CFD), Finite Element Method (FEM), and multi-body dynamics are now combined within aeroelastic design frameworks for full aircraft configurations [81]. However, considering the finite computational resources and the strong correlation between analysis fidelity and the required computational time and cost for that level of fidelity, compromises are usually necessary between the level of fidelity and the scope of design space. Regardless of the number of coupled disciplines, adoption of high-fidelity, physics-based tools fundamentally constrains the

amount and the range of design variables limiting the overall validity of MDO methodologies to the restricted design space only. In this way, current MDO practice serves as a mean to *design improvement* rather than as a *design optimization* framework.

On the other hand, driven by the emphasis on environmental impacts and ever diversifying performance needs, conventional designs are rapidly being replaced by unconventional ones which can only be realized through highly expensive M&S procedures and extraordinary know-how. Designs with wide variety of design variables encompassing multiple topologies and situations where a large number of design variables must be explored concurrently are becoming more typical for the design of advanced aeronautical systems, especially in the early phases of their design processes. For these cases, the establishment of a well-posed design problem and the determination of unbiased and effectively-focused design space are very challenging tasks.

This gap between the finite computational resource and the challenging level of design complexity is the birthplace of surrogate modeling. For the past few decades, surrogating computationally expensive analyses inside the higher-level design optimization loop has proven to be a powerful enabler to extend opportunities in MDO approach.

The ongoing evolution of surrogate modeling techniques recapitulates the general evolution of regression analysis methods. Since the early period of surrogate modeling using linear regression models such as the response surface method (RSM), various adaptive regression methods including neural networks and kernel methods are now frequently applied in the field of surrogate modeling. The main driver for this evolution from linear models into adaptive methods is to lessen the computational burden of the scalability problem as the dimensionality of the problem escalates. Linear regression models using pre-defined basis functions suffer *'curse of dimensionality'* as

the number of design variables increases due to their formulation itself directly coupling model parameters with the design variables. Practically, based on the Pareto principle and the ANalysis Of VAriance (ANOVA) techniques, concentration to a limited number of design variables can diminish this hurdle, but the complex problems with intrinsic nonlinearity and interaction between large numbers of design variables are beyond this remedy. Adaptive regression models are much more useful in this respect because these models primarily depend only on training data regardless of the number of design variables. Hence, to facilitate practical-scale problems having a larger number of design variables, adapting the basis functions to the data themselves is necessary, rather than linearly combining the fixed basis functions. Two realistic ways of doing this are either defining basis functions centered on the training data points or allowing fixed number of basis functions to adapt their parameters to the training data. The first approach resulted in the kernel-based methods such as Gaussian processes and radial basis function networks and the second approach corresponds to the Multi-Layer neural networks such as Multilayer Perceptron (MLP).

The kernel based methods add basis functions for each and every training data example. This memory-based nature, which stores training data for prediction phase, makes these types of methods quick to train but slow at prediction. To overcome this shortfall, the more sophisticated kernel methods that use only subset of training data, such as Support Vector Machine (SVM, [61, 84, 9]) and Relevance Vector Machine (RVM,bishop2,tipping), have been developed toward the more compact models depending on the sparser training data. On the other hand, the neural network methods use parametric forms for basis functions whose values are adapted only during the training phase. Therefore, while the training phase requires considerable computational efforts, the neural network methods usually result in a more compact functional form and more rapid prediction than the kernel-based methods. Because the numerical efficiency in surrogate modeling is paramount, especially in prediction phase,

neural networks are usually preferred in many surrogate modeling tasks for the modeling of physics-based analyses. Moreover, there is one more important reason which makes neural networks popular in surrogate modeling tasks related to the nonlinear mapping capability between input and output of training data. As shown in many mathematical texts (including [9]), the kernel functions can be derived directly from the linear models and there exists mathematical duality between kernel functions and weight parameters in the regularized least square method, meaning that kernel methods and linear regression models share the fundamental mathematical characteristics. While the linearity in the way of combining basis functions in linear models does not necessarily limit the applicability of linear models within the linear relationship in input-output mapping, the neural network is known for having the relative advantage in the aspect of representation capability for nonlinear relations between inputs and outputs of arbitrary training data. Therefore, a surrogate modeling using neural networks is an important research field to overcome the challenge for the efficacy of the MDO strategy.

1.2 Artificial Neural Network

Artificial Neural Networks (ANNs) are information processing constructs made up of simple computational units inspired by the neural structures in the biological Central Nervous System (CNS). The general intention of the ANN research and application is to emulate the unique information processing capability of the animal brain, including learning ability, by abstracting its components, structures, and mechanisms. McCulloch and Pitts' classic paper in 1943 [58] on a logical calculus of ANNs that unified the studies of neurophysiology and mathematical logic showed another way for general computation based on the simple 'all-or-none' activation function and this paper is generally agreed upon as the birthplace of the disciplines of the ANN.

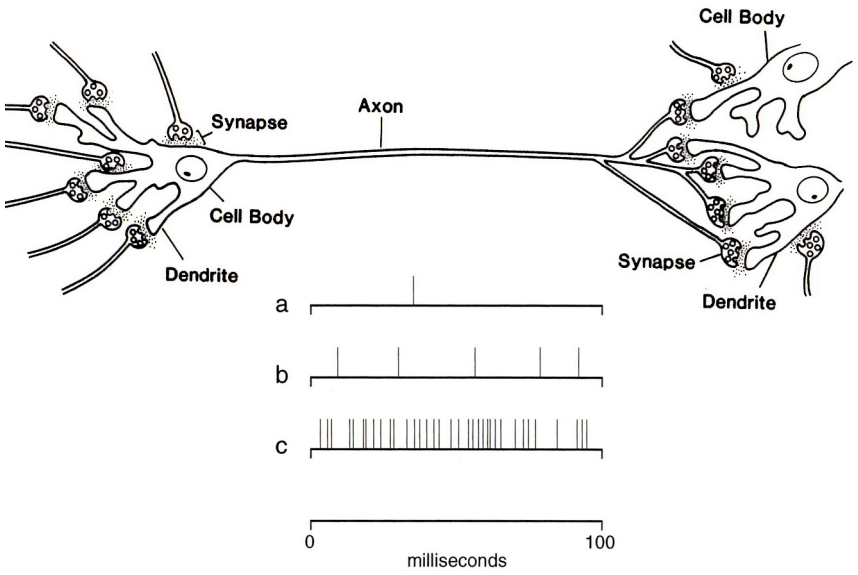


Figure 1: A schematic diagram of a *"typical"* vertebrate neuron and the firing of a single neuron [17]

Although many different types of ANN architecture and more sophisticated artificial neuron models have been and are being developed, the fundamental theories and applications of the ANN have grown out of two concepts;

1. The simple artificial neuron model with nonlinear activation function
2. The training process modifying memory of network using synaptic strength (or connection weight) changes

The synergy between these two simple concepts, remarkably, has been found to provide nonlinear input-output mapping capability in an adaptable and fault-tolerant manner. Note that the biological neuron's typical behavior is to fire or not through the axon depending on its inputs arriving at the dendritic branches (Figure 1). In case of firing, it fires at different levels of frequency and it is believed that the stronger the stimuli in the inputs, the faster it fires. The nonlinear activation usually that employs sigmoid-type functions with the adjustable weight parameters in the current convention of the ANNs is only loosely related, at best, to this *'all-or-none'* firing with

variable frequency in the biological neural network although there exist more sophisticated artificial neuron models such as dynamically firing, time-dependent neuron models.

1.2.1 How Does It Work?

A basic calculation ability of the ANN comes from setting the weights in such a way that when the input units are given certain values, the output units are activated appropriately. This means that a mapping between inputs and outputs is achieved and hence that a function can be executed, which is the basic definition of the calculation. Therefore, to build a properly working neural network, one has to decide: how to set the weights, whether they are modifiable, and if so how and what range of activity values a unit may take and how they are determined, how to represent the input vectors, and the nature of the connectivity between computational units [15].

In other words, the ANNs do not operate by being programmed; they *learn* by being trained. The *learning* is the adjustment of the network in response to external stimuli and the *training* refers to the presentation of the inputs and target outputs (in case of supervised learning) to the network [63]. In general, training, and hence learning, is just the means to an end. After training (and learning), the ANNs generalize outputs for the newly given, unseen inputs by using memories stored within its connection structure and weights.

The conventional learning paradigm of the neural network focuses on how to set the weights assuming other network design decisions are set a priori. Noyes described this as follows [63]; *"... the term 'learning' applied to neural networks usually refers to learning the weights, ... This definition excludes other information about the network that might be learned, such as the way in which the neurons are connected, the activation function and parameters that it uses, the propagation rule, and even the training rules themselves."*

In a working network, the input units feed the input signals to the other computational units (such as the hidden units and/or the output units) and the output units deliver the prediction as the output signals. In the training phase, the hidden units develop weights that recognize particular features from the input signals, working as internal sub-tables if an entire neural network being a table look-up process, and deliver this processed information for the other hidden units and/or the output units. In the example case of a sonar signal classification task, discriminating whether a particular sound signal is reflected from a rock or a mine, the hidden units developed weights that recognize, primarily, the frequency bandwidth, the signal onset time, and the rate of decay of the signal [15].

This particular method of calculation has a great advantage when the massive *table-look-up* tasks are important whose entries are stored as prototypes which have to be learned through the data. It is because a network, by its architecture itself, executes the sequential components of such tasks, i.e., storing entries, making the distance measure, finding the match, and delivering answer in a unique, very efficient manner within only a few steps [15]. Recently, the ANN is widely applied in the fluid mechanics field [26]. Extremely accurate time-dependent predictions were obtained for a physical system dominated by three-dimensional unsteady fluid mechanics [25]. ANNs also were shown to be highly effective for fault diagnostics and control reconfiguration [31, 95]. In addition, ANN-based control systems were shown to produce accurate results for complex flow control systems [76, 83]. All these outcomes strongly indicate that the ANNs are a very useful means for addressing nonlinear, time-dependent physical systems. Following Faller and Schreck, the particular strengths of ANNs relevant for fluid mechanics applications are as follow [26];

1. A significant strength of the ANN approach is the relative ease with which fully coupled nonlinear input-output mappings can be calculated.

2. Further, the form of the nonlinear equation system does not have to be known in order to derive a nonlinear model of a physical system.

3. The difficulty associated with deriving an ANN solution typically decreases as the number of degrees of freedom increases.

4. While ANNs require quantitative experimental data and/or computational solutions, only limited data are required to determine a solution which is accurate throughout the parameter space.

5. The final solutions are computationally very fast - milliseconds in software and microseconds in hardware.

The Multilayer Perceptron (MLP) is one of the most popular ANN architectures, where neurons are grouped in layers and only forward connections exist between adjacent layers. This proves a powerful tool for prediction with advantages such as nonlinear mapping and noise tolerance. The interest in the MLP was stimulated by the advent of the Back-Propagation (BP) algorithm in 1986 [74] and since then, several rapid variants have been proposed [71]. In the literature, the term, error back-propagation, is used as either a weight-error gradient (the derivatives of the error function with respect to the weights) calculation algorithm or a complete learning algorithm. In general, a learning or training method for the ANN has two distinct phases, the first phase of estimating the present network's status and the second for adjusting network to the next state. The BP as a complete learning algorithm advances to the next weight set by adjusting weights in similar way to the steepest descent method involving a learning factor that determines the abstract size of each step of adjustment while the BP as a gradient calculator is usually harnessed by another outer-level weight optimizer that operates depending on the gradient information provided by the BP algorithm. In the latter case, potentially any gradient-based optimizer could be used as the second phase of the training method. Due to its

numerical simplicity and efficiency, the BP is an almost unanimous choice for the weight-error gradient calculation algorithm. Among the efficient gradient-based optimization algorithms such as conjugate-gradient method and Newton's method, the Levenberg-Marquardt (LM) algorithm is the most popular match to the BP as a gradient feeder. Since the early application of the LM method into the ANN weight optimization in mid 1990s [34], the 'MLP/BP/LM' framework became one of the most popular 'architecture/gradient calculator/weight optimizer' combinations in practical ANN applications due to its numerical efficiency for the function approximation tasks. This particular ANN framework is implemented in many commercial computation software packages. For example, the default setting in Matlab's ANN toolbox builds an MLP structure depending on the user's selections such as the training data and the number of hidden units and, then, trains the network with the BP-LM algorithm pair.

1.2.2 A Bigger Picture

Rojas categorized the general ways of computation into five representative models [73]. Since the computability itself had been formally defined by several mathematicians, the computer model based on von Neumann architecture has made great stride taking the central stage of computation among these five models; mathematical, logic-operational, computer, cellular automata, and biological model. Here, the biological model means the ANN. ANNs have a hierarchical multi-layer structure which sets them apart from cellular automata and they do not operate sequentially by pre-assigned programs, as Turing machines do.

But the current practice of an ANN implementation is, in general, a kind of emulation only in the software level, constrained within the fundamentally sequential, conventional digital computers. The real potential can only be realized when an ANN

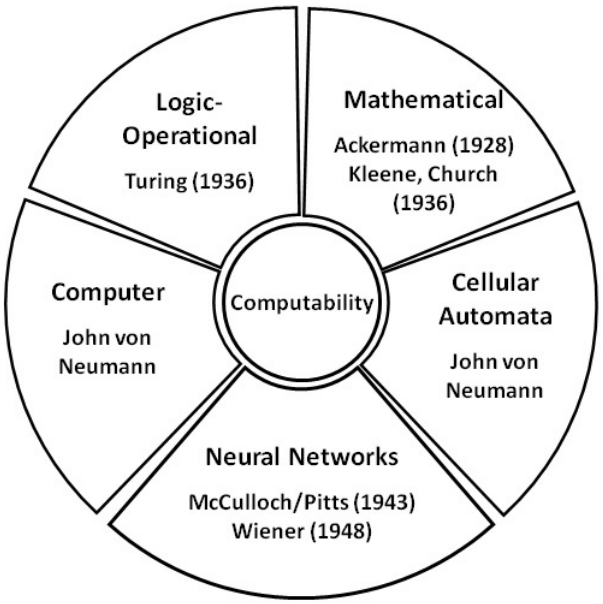


Figure 2: Five models of computation [73]

is implemented on the hardware level enabling true parallelism intrinsic to the structure of the ANN. At present, many research entities are finding ways to implement a more brain-like computational hardware. By rethinking the fundamental balance between information, noise, and energy, the new silicon chips are being built to achieve computation capability with radically reduced energy consumption in the field of neuromorphic engineering [14, 10]. The Defense Advanced Research Projects Agancy (DARPA) and the IBM corporation recently announced the manufacturing of the two prototypes of silicon chips whose internal arrangement is designed mimicking synaptic connection structure of the neural network. One of these chips are known to recognize input patterns based on the reconfigurable connection weights and connection structures [45]. The Ecole Polytechnique Federale de Lausanne (EPFL) and the IBM corporation are also investigating the possibility of digital simulation of the way animal brains work. They reported on the success of construction of the hardware-level neural network simulating a rat brain using supercomputing technology [56].

1.3 Designing Neural Networks

The pursuit of efficient neural computing has been motivated by the belief that many of engineering challenges being encountered during the M&S tasks in aeronautics could be eased by this particular type of approach. The growing number of ANN applications into the fluid mechanics field supports this belief. The ANNs are frequently used as the means to surrogating the high-fidelity CFD analysis codes. They are also used to enable stochastic aerodynamic shape optimization tasks many of which have been computationally prohibitive with the brute-force analyses. Many researchers in aeronautics nowadays apply the ANN techniques to the aerodynamic shape optimization tasks such as for airfoils, aircraft wings, and turbo-machinery components resulting in promising outcomes [65, 22, 66, 67, 33, 43]. However, referring to these works, the significant portion of research task had to be devoted for finding the suitable ANN structure prior to or iteratively with the main implementation of their optimization methods. And this subtask is done in an exhaustive trial-and-error manner. For example, the most popular choices of ANN architecture such as Multilayer Perceptron (MLP) or radial basis function network (RBFN) require the determination of the number of hidden layers and the number of hidden units in each layer or other global parameters for basis functions. At present, while there exist many available methods (e.g., back-propagation method or generic algorithm) for optimizing network connection weights for a given structure of ANN, there is no well-developed theory for the determination of the architectural parameters and, more generally, for optimizing the connection structure of the ANN. Haykin emphasized that *"the satisfactory answers to these issues are usually found through an exhaustive experimental study, with the designer of the ANN becoming an essential part of the learning loop [37]."* This problem imposes an evident and significant gap for the ANN surrogate modeling considering that the structure (topology, morphology, or architecture) of the ANN has long been known as the critical factor determining ANN's generalization

performance [43, 80, 93].

1.3.1 Existing Methods

Aside the detailed internal connection structure, the size of the networks of a similar topology alone has been known for its fundamental impact on the generalization performance [51, 71, 13]: Too small a network cannot learn the problem well, but too large a size will lead to over-fitting and poor generalization performance. To properly size an ANN, two groups of methods have been developed.

- Constructive algorithm adds hidden neurons to the current ANN model in case of expected performance improvements. Once the hidden neurons have been added excessively, they cannot be suppressed to reduce the size of ANN, thus resulting in unwieldy ANNs, and, without validation or early-stopping mechanism, adding new units eventually lead to over-fitting.

- Pruning algorithm eliminates neurons of the current ANN model probing the possibility of improved performance. Normally, the ANN must be retrained after pruning. If the selection criteria, determining the particular neuron that will be removed, are guessed correctly, it is possible to obtain a better solution.

Finding appropriate size of network by growing or pruning can result in the network providing better generalization, but these hill-climbing methods, aside the susceptibility to be trapped in one of the local optima [72], have significant limitations to be applied to general ANN architectures. Some of these methods have been developed for particular types of network architectures with specially devised learning methods. Others of these methods require frequent re-training procedures after additions and eliminations of computational nodes and/or connections. The detailed reviews on these methods are provided in the next chapter.

An alternative approach is to optimize both the structure and weights of the networks using evolutionary algorithm which performs a population-based global search

such as Genetic Algorithm (GA) and Genetic Programming (GP). This approach is the main theme in the field of the neuroevolution and often called as the TWEANN (Topology and Weight Evolving Artificial Neural Network) method. The primary concern in this approach is the significantly escalated computational time required to train entire generations of populations consisting of large number of individual networks each of which has to learn the optimal weights. The applicability of these methods to surrogate modeling tasks for the physics-based analysis that involves large number of input and output variables is, at best, very limited.

Therefore, the lack of an efficient network design method has not yet been completely filled by above three approaches; the constructive methods, the pruning methods, and the neuroevolutionary approach.

1.3.2 Scalability Problem

The training time of the MLPs does not scale well and it is very slow in networks with multiple hidden layers [41]. Due to the nature of the MLP architecture, which assumes fully connected adjacent layers, even small increments in the number of input parameters, output parameters, hidden units, and/or hidden layers cause large increments in the number of total network connections requiring exponentially increased computational time and the memory usage in the weight optimization process. This scalability problem of the MLP within the conventional learning paradigm imposes a fundamental limitation on the applicability of the ANN surrogate modeling. Contrast to the kernel-based regression methods which are evolving toward the more efficient, sparse kernel methods such as Support Vector Machine (SVM) and Relevant Vector Machine (RVM), the ANN methods are, in some sense, *stagnated* at the MLP architecture in the aspect of the scalability problem. The ability to access the more suitable and significantly sparser connection structure than the MLP can be an effective remedy for the scalability problem extending the applicability and numerical

efficiency of the ANN surrogate modeling. In this context, a principled methodology for the structural design of the ANN is a key enabler to the scalability problem. The necessity to overcome the current gap has also been clearly expressed by Hüsken et al. In their research case directly dealing with the structural optimization of the ANN for its proper application to stochastic optimization of turbine blade using CFD, they stated that; *"The performance of ANNs does not only depend on the choice of the weights, but also strongly on the choice of the architecture or structure, i.e., the graph describing the number of neurons and the way the neurons are connected. In particular, the task of fast learning or learning with a small amount of data demands a suitable structure." [43]*

The primary objective of this thesis is to investigate the main source of difficulty in the neural network design and to seek an effective and meaningful solution to reduce that difficulty.

CHAPTER II

RELATED WORKS

In this chapter, a brief literature survey is described for two of the most relevant issues on improvement of the ANN surrogate modeling. The first is the current practice of the ANN surrogate modeling in the field of aerodynamics. This is about how computationally expensive CFD analyses are surrogated via the ANNs and what are the primary challenges in doing those tasks. The main difficulty lies in the selection of appropriate architecture and size of the ANNs. Hence, the second literature review is for the currently available network design methods. In general, these methods can be grouped into one of the three classes; the constructive methods, the pruning methods, and the evolutionary approaches. The underlying ideas for each method and their limitation in the context of the surrogate modeling tasks are discussed.

2.1 Surrogating CFD Analyses via Neural Network

Practical needs for surrogating CFD analysis come from two reasons. The first is purely from the high computational cost of the CFD. Bernstein et al. summarized this problem as follows [6]; *"Despite the rapid development of computers and computational techniques, CFD analysis of aerodynamic characteristics is still time-consuming which hampers aerodynamic design of a large number of aircraft layouts."* The more delicate second reason is that every CFD analysis requires sophisticated tasks related to the generation of the computational grids. Converting time-consuming calculation into a compact and fast surrogate model which directly maps the geometry of the object or aerodynamic component itself and the flow conditions to the aerodynamic characteristics without the need for complicated grid generation procedure is a common goal of surrogate modeling for the CFD analysis.

2.1.1 Stochastic Aerodynamic Shape Optimization

The most frequent and important targets of the surrogate modeling of physics-based analyses are in the design optimization tasks replacing the computationally expensive original analysis modules. In particular, the most popular applications of the ANN surrogate modeling for the CFD analysis are found in the stochastic aerodynamic shape optimization which is based on the following two characteristics.

First, the stochastic optimization methods such as Genetic Algorithm (GA) and Simulated Annealing (SA) use population-based heuristic search, which attempts to find global optimum without the necessity of gradient information which is susceptible to be trapped near local optima. However, the required number of function evaluations during the stochastic optimization process is much higher than that number of the deterministic methods. Second, the ANN is capable of complex nonlinear mapping between input-output parameters using relatively scarce training data in a noise-tolerant manner, and acquisition of an ANN surrogate for high-fidelity, physics-based analysis tool is practically feasible task. By partially replacing computationally expensive high-fidelity, physics-based analyses with ANN surrogates, the stochastic optimization process can be accelerated reducing required computational time and cost.

Therefore, the primary value of the ANN is to construct an accurate and reliable surrogate model, which makes the most use out of each and every computationally expensive CFD run and to accelerate design optimization process.

Rai [68, 66, 67] extended polynomial surrogate models using ANN's prediction capability for aerodynamic shape optimization using CFD tools. In his research, many important concepts on adaptive redefinition of design space, adaptive training, and unique ensemble method to augment extrapolation performance of ANN have been investigated.

Duvigneau and Visonneau [22] performed stochastic aerodynamic shape optimization for the two-dimensional airfoils and the three-dimensional wings using GA optimizer. To enable efficient and accurate evaluation of aerodynamic performance, they trained ANN surrogate and replaced computationally expensive CFD runs by trained ANNs. By the trial-and-error approach they found the enough number of CFD runs to train radial basis functions networks (RBFNs) and used this information at each generation of GA population to reduce the number of CFD runs. Compared to the direct combination of GA and CFD runs, the significant reduction in computational time was resulted in by this approach and the interestingly unconventional optimum was obtained for the wing shape. In this research, the proper ratio between CFD runs and ANN predictions which governs the overall optimization efficiency had to be found in the trial-and-error manner. And the large amount of numerical tests had to be executed to determine the suitable size of the ANN and the network structural parameters such as the attenuation coefficient.

Hüsken et al. provided the method for optimizer to automatically determine which of either the exact CFD run or the ANN prediction has to be executed by the numerical measure of their performance by the principled approach, i.e., the higher the quality of the model is, the more often it should substitute the original fitness function [43]. They also noted the difference in the required ANN performances for dissimilar applications; for a surrogate model exploited for higher-level optimization procedure, a qualitative approximation is often sufficient to fulfill the task, whereas a surrogate model for prediction needs a minimal quantitative difference. This means that the surrogate modeling might require rather relaxed prediction performance which only enough to track the qualitative variation of objective function instead of higher numerical precisions required for stand-alone prediction cases.

Hacioglŭ [33] also combined the GA and the ANN but with different strategy to overcome the common problem of prohibitive computational time involving CFD

analyses. He had trained Multilayer Perceptron (MLP) network for airfoil pressure-to-geometry mappings for the inverse design problem of the airfoils and made this ANN to predict elite offspring which produces the closest pressure distribution to the given target for the next generation instead of more conventional ANN's usage as surrogate model. This approach also brought significantly accelerated convergence on GA operations.

Pierret and van den Braembussche [65] solved the inverse design problem of turbine blade using Simulated Annealing (SA) with the Navier-Stokes solver. The ANNs were trained for complex mapping between the geometric parameters of the turbine blades and their aerodynamic characteristics. In this case, the ANN was used to replace CFD runs during the optimization process using the SA. With the small number of Navier-Stokes computations an optimized blade satisfying both the aerodynamic and mechanical requirements was obtained. In this result, the use of the ANNs was critical to synthesize aerodynamic data from the previous designs and to reduce the required CFD analyses.

2.1.2 Aerodynamic Brain Approach

Several researchers probed the possibility of making problem-specific *'artificial brain'* using the ANNs. Here, the emphasis is on the quantitatively accurate prediction of aerodynamic characteristics as a stand-alone estimator. In this case, numerical prediction accuracy becomes more important than the surrogate modeling plugged in the stochastic optimization loop where the qualitative representations of those characteristics are often sufficient requirement to guide the design process until the pseudo-global optimum is reached. Figure 3 depicts this point of view where, although the approximation errors of the ANN model are quite non-negligible, the optimization using this model will lead to the desired local minimal fitness.

Faller and Schreck performed notable researches for ANN modeling of unsteady

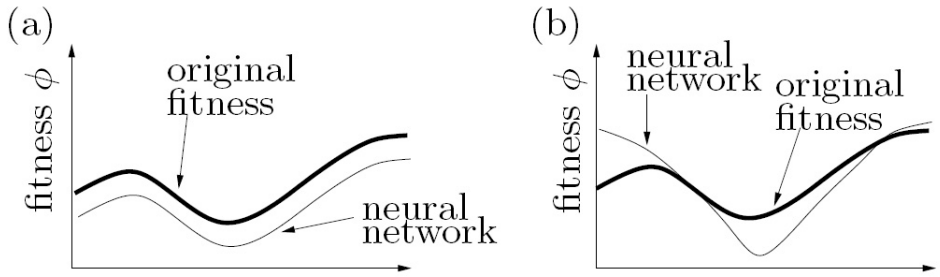


Figure 3: Quantitative and qualitative modeling requirements [43]

fluid mechanics [25]. With relatively simple ANN structure, they succeeded to predict the fluctuation of surface pressure in multiple points on the pitching wings. Considering very limited training domain compared to the wide variety of the experimental conditions, the accuracy of their result is quite remarkable showing that the great potential of ANNs in the analysis of complex unsteady aerodynamic phenomena.

Hazarika et al. reported a grounding research on the mathematical mapping between the geometry and the surface pressure on the airfoils in the subsonic flow regime [38]. They performed both the airfoil geometry-to-pressure coefficient mapping and the pressure coefficient-to-geometry mappings using the single-hidden-layer MLP network. They determined the size of the training set and the number of hidden units based on the empiricism. To surrogate target airfoil pressure coefficients obtained using a panel method varying the angles of attack and airfoil geometry defined by dozens of point coordinates. Figure 4 shows the network architectures which used for the pressure coefficient-to-geometry mapping (left) and the airfoil geometry-to-pressure coefficient mapping (right) respectively. For both architectures, they used 10 hidden units.

They achieved virtually identical modeling capability to the original target data in the both modeling cases (Figure 5). They also probed the noise-tolerance modeling capability of their ANN models by artificially adding noise terms in pressure coefficients obtained from the panel method simulating the measurement error which

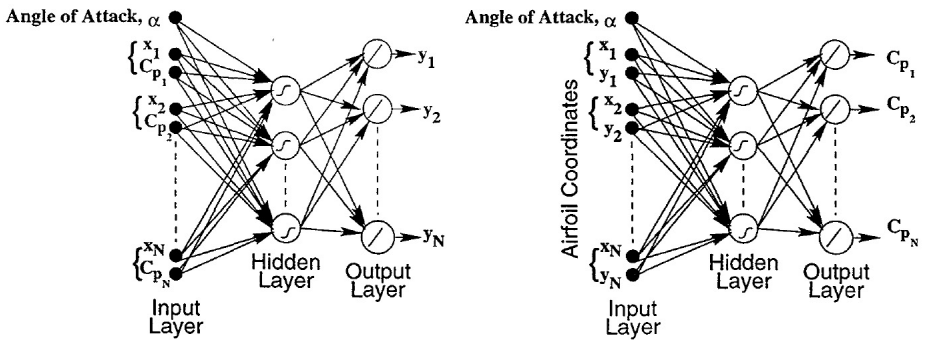


Figure 4: The MLP models used in the airfoil inverse design (left) and the airfoil geometry-to-pressure approximation (right) [38]

prevails in the raw data acquired from the wind tunnel tests. Figure 6 confirms that the ANN follows the original panel method results despite of the added noise in the target pressure coefficients. They emphasized the meaning of this type of generalization characteristics as; *"Note the excellent generalization capability of the model. This very important result indicates that it may be possible to utilize noisy data obtained from wind tunnel experiments for design purposes, using pattern recognition based algorithms such as artificial neural networks."*

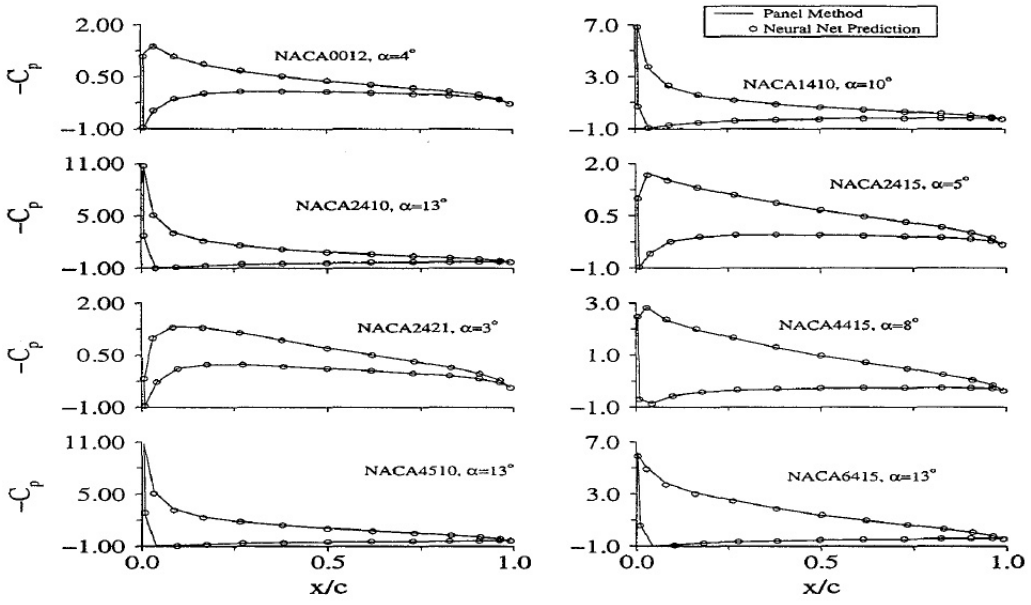


Figure 5: Computed and predicted pressure distributions [38]

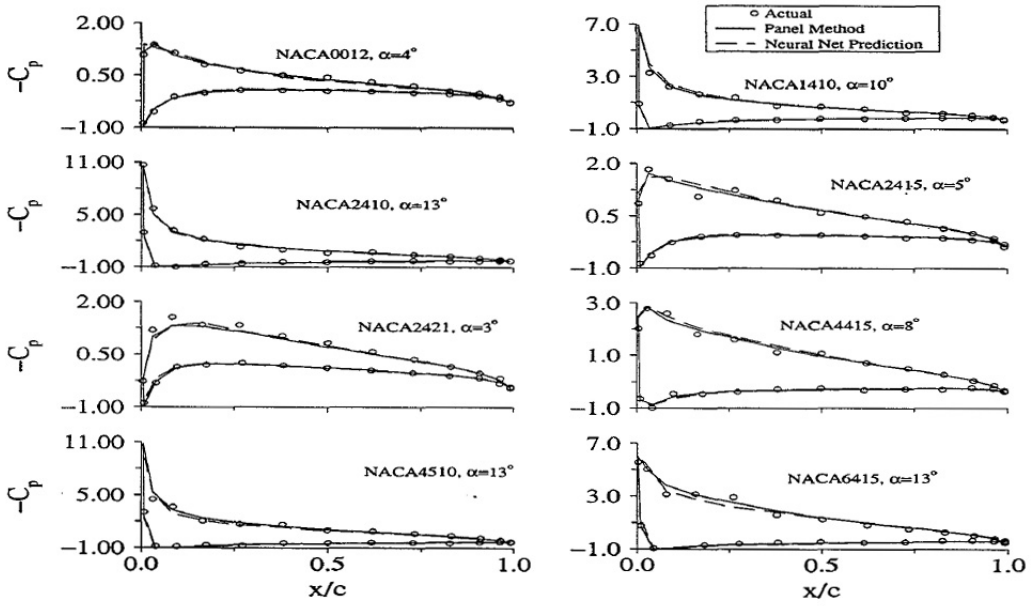


Figure 6: A comparison of the pressure distribution computed by the panel method and that predicted by the neural networks for various flow conditions and airfoil contours using simulated noisy data [38]

Da Silva et al. trained an ANN with systematic data consisting of wing geometry-to-pressure distribution mappings for a particularly shaped class of aircraft wings [20]. They trained two-hidden-layer MLP having 20 hidden units in each for the target pressure differences on the main wing surface of the jet transport aircraft. As a preliminary study, they tried to surrogate the CFD code, BLWF (Boundary Layer Wing-Fuselage), for the wide range of wing/section geometric parameters and flow conditions encompassing subsonic and transonic flow regime requiring tens of thousands of training data points. Their result shows the difficulty of accurately modeling shock waves in the transonic flow regime when the typical MLP architecture is used. Figure 7 shows three example cases of that difficulty.

In these cases, the ANN wrongly predicts location or intensity of the shock, or smoothes out the shock wave completely. This modeling inefficiency related to the shock wave is one of the fundamental challenges in the transonic aerodynamic modeling distinctive from the subsonic cases.

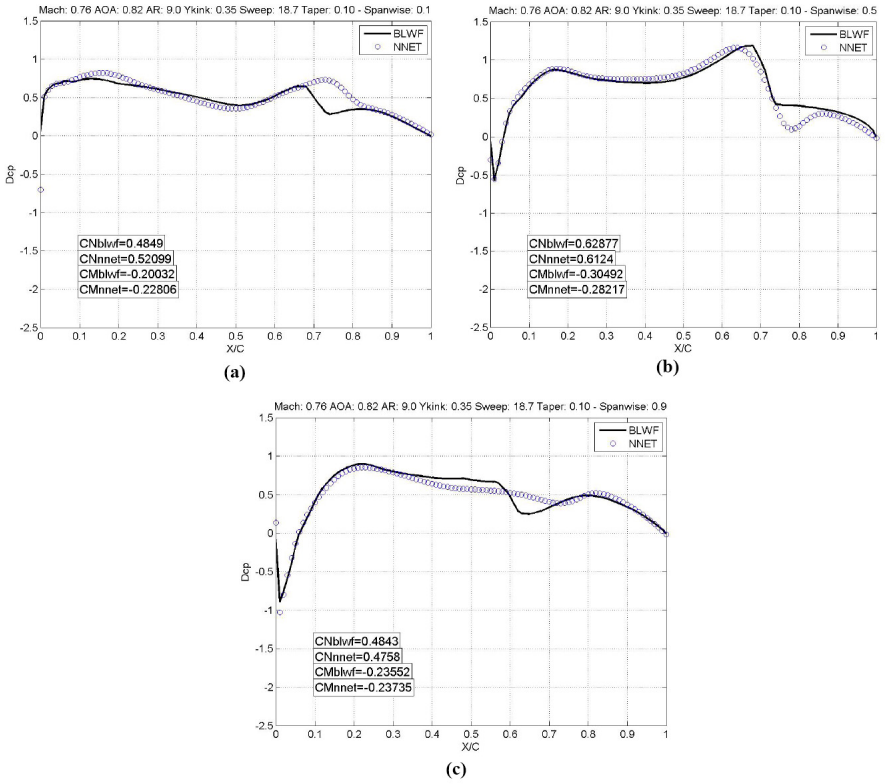


Figure 7: High transonic flow with positive angle of attack on the root (a), half-wing (b), and tip (c) station [20]

Theoretically, two remedies can be tried; increasing the amount of the training data which include more cases of shock phenomena or adjusting network architecture which is more capable or suitable for capturing discontinuity caused by shock waves. But considering the large amount of computational time required for the ANN training, the additional expansion of training data set is hard to be implemented. Under the current MLP conventions, adding more hidden units can cause large increment in the number of network connections and, hence, again, significantly more network training time and cost. Their separated construction of the training sets for the subsonic-to-low transonic model and the high transonic model resolved the discrepancy in the shock prediction, and they concluded with following comments which reflect these difficulties [20]; *"Even though a more complete study is necessary as to finding the best neural network architecture for each analysis, this work managed to show that promising results were obtained using the proposed methodology ... Studies shows that the computational cost required for training the neural network increases linearly with the training set size and exponentially with the neuron numbers. As mentioned earlier, the case study considering planform and airfoil variation implied on a very expensive computational cost, hence, alternative solutions for neural network training shall be considered."*

Dos Santos et al. reported similar problem for geometry-to-aerodynamic coefficient mappings of two-dimensional airfoils [21]. In this case, the CFD code, MSES, has been surrogated via the MLP networks for the sectional lift and drag coefficients. They performed the sensitivity study using multiple MLP structures adjusting number of hidden units and one of the typical result shown in Figure 8 indicates the significant discrepancy between the ANN model and the CFD result as the flow Mach number increases.

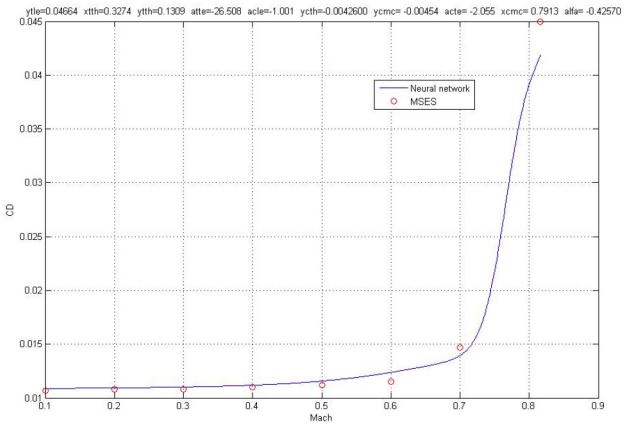


Figure 8: Predicted and calculated C_d for one airfoil in the validation set [21]

2.1.3 Challenges in the Surrogate Modeling of the CFD Analysis

Summarizing the previous sections, the applications of ANN surrogating of CFD analyses have been pioneered largely in three directions.

1. To reduce the required number of computationally expensive function evaluations involving CFD runs in each generation of the stochastic optimization process, from the second generation, a particular portion of exact function evaluations is replaced by ANN predictions and usually all the CFD runs are used to train ANN.

2. To reduce the required the absolute number of generations for the stochastic optimizer to converge, from the second generation, the elite offspring based on the ANN's prediction are inserted to accelerate evolution process of population.

3. A problem-specific network is generated using input-output mapping data generated by CFD runs. Then this ANN is used for inverse design procedure or a general meta-model of the CFD analysis.

In general, majority of these applications of ANN surrogate modeling to aerodynamic shape analyses show the great opportunities for computationally challenging

aerodynamic M&S tasks to be tackled by ANN computation paradigm, especially, making previously computationally prohibitive stochastic optimization practically feasible. Majority of researchers adopt Multilayer Perceptrons (MLPs) or radial basis function networks (RBFNs) as ANN architecture but some of them provide incomplete justification about their choices of the particular ANN structure.

In case of the MLP applications, while the number of input and output units are determined by the given task, the other structural parameters such as the number of hidden layers and the number of hidden units in each hidden layer are not conveniently determined by the given task. And in many cases of RBFN applications whose network structures are simpler than MLP, the one of the reasons for that choice of RBFN instead of MLP is due to the difficulty to determine the architectural parameters for MLP although there are additional or unique parameters need to be determined for the RBFN such as attenuation parameter [22]. Several authors provide sensitivity analysis results for handful variations of the number of hidden layers and/or the number of hidden units with separately devoted computational experiments to their prediction capability.

Largely, the proven generalization capability of the ANN is confined to the cases where the training set is bigger than enough and the underlying relationship between inputs and outputs is simpler than the ANN's generalization capability. But the practical challenges in the MLP applications occur when the ANN model fails to provide satisfactory prediction capability. The degraded generalization capability in the transonic flow regime is one such example. Due to the very nature of the CFD analysis being computationally expensive to expand to a big enough set of training data, and due to the very nature of the MLP architecture consuming exponentially increased training time and cost as a single hidden neuron is added to increase the generalization capability, we have to seek a solution by enhancing training efficiency which tightly coupled to the connection structure of the ANN and its training algorithm.

2.2 Impact of Topology

Many researchers believe that the architecture of the ANN has the critical impact on its generalization capability [43, 80, 93]. Simply speaking, considering a generic network made of nodes and links, such as the internet, an electrical grid, or the brain, two such networks with the same number of nodes and links but with different connection structures can display quite different behaviors and the role of this network architecture becomes even more important when its components are adaptive, such as in the ANNs where connection weights and the threshold (or bias) values are adjusted according to the activity of units they are connected to [28]. In this case, the connection structure affects both what functionality is acquired by the network during the process of change and how the network behaves after that process. On the other hand, a universal approximator theory [19] exists saying a fully connected, single-hidden layer network can, in principle, approximate any continuous function. This section attempts to investigate the impact of the ANN architecture for the practical surrogate modeling tasks.

2.2.1 Model Selection Problem

The relationship between general model architecture and its modeling performance has usually been investigated in the model parameter space. For example, in linear regression models with particular type of basis function, the order of this basis function controls the number of free parameters and thereby governs the model complexity. And the determination of the number of free parameters is called the model selection problem [9]. In this context, model complexification means the increase in the number of model parameters which, in excessive case, fundamentally results in over-fitting problem and, therefore, the key issue is the proper regularization mechanism for the best generalization performance. The pure maximum likelihood approach results in

over-fitting in case of using excessive number of model parameters without appropriate regularization or validation mechanism while the Bayesian approach which maximizes posterior distribution intrinsically regularized model by itself and does this fundamentally depending only on the training data alone without the need of the additional parameter such as the regularization parameter in maximum likelihood approach. In a fully Bayesian approach, penalties for complexity arise in a natural and principled way. In this aspect, the Bayesian treatment is figuratively compared to Ockham's Razor [55]. And for given structure of ANN. a fully Bayesian approach can provide automatically regularized weight parameters which can generalize better outside the training domain but this comes with the expense of computationally demanding marginalization over the distribution of weight parameters. Thus, for the given structure of the model, the complexity of the model has already been determined by the given model structure itself, e.g., the order of polynomial basis function, the number of connection weights in ANN, etc. In case of excessive complexity, regularization is required to guarantee the generalization performance out of the training domain. The way to obtain optimal generalization performance depends on how to penalize the excessive complexity; maximum likelihood approach needs explicit regularization with independent process for determining regularization parameters while the Bayesian approach does this implicitly at the expense of marginalization of probability distribution of model parameters.

In the ANN context, many dissimilar models having different connection structures exist with the same number of the weight parameters. Extending the degree of freedom for model selection problem beyond the parameter space might require rather different strategy to obtain the best generalization performance because, by modifying the model structure (or architecture, or topology) directly, the complexity of the model can be adjusted by other means than just the number of the free parameters in a model. In this case, the maximum likelihood approach for minimum

training error might or might not produce over-fitting just depending on the number of free parameters. Because the necessity of regularization comes from the uncontrollable model structure, optimizing both the model structure and parameters faces completely different situation. One of the formal mathematical investigations of the relationship between ANN's architecture and its function resulted in the following conclusions [53, 35];

- Each combination of the particular ANN and the particular learning process has unique 'probability distribution of network configuration' over the space of its possible input-output mappings and the shape of this distribution is an intrinsic property of the two;
  - The architecture of the ANN
  - The dynamics of learning for the ANN
- The entropy of that distribution is defined as a measure of the diversity of the possible input-output mappings by the ANN.
  - Supervised learning can be viewed as a method to reduce the intrinsic entropy by excluding network configurations, i.e., particular combinations of connection weight parameters, that realize mappings incompatible with the training set.
  - Effective rule extraction from examples must be directed at finding mappings compatible with the entire task domain, rather than just with the training examples.
- Because 'probability distribution of network configuration' and thus the functional capabilities of the network are determined both by the architecture of the ANN and by the dynamics of learning for the ANN, they can be influenced by two ways;

– By changing the dynamics of learning, e.g., the development of more efficient learning algorithms which have been limited to the adjustment of small number of relatively fixed formalisms like delta-rule learning and Hebbian learning.

– By changing the architecture of the ANN, which is much more versatile method of changing the 'probability distribution of network configuration.'

Hence, these investigations provide the theoretical justification for the structural exploration of the ANN to enhance the generalization performance of the ANN. Happel and Murre also pointed out the difficulty of proper architectural change as follows [35]; "A major problem with this approach, however, is that there currently exist no general methods or guidelines providing useful architectural constraints. This may be one of the reasons why entropy reduction was largely left to a learning process in unstructured networks.". Here, 'learning process' means the conventional paradigm of ANN training which only optimize weight for a given connection structure and 'unstructured networks' corresponds to the fully or regularly connected multi-layer network architecture.

2.2.2 Kolmogorov's Theorem and Cybenko's Theorem

Kolmogorov showed that every continuous function of several variables can be represented as the superposition of a small number of functions of one variable [50]. Particularly, in the context of ANN, any continuous mapping $y(\mathbf{x})$ from d input variables x_i to an output variable y can be represented exactly by a three-layer ANN having $d(2d+1)$ units in the first hidden layer and $(2d+1)$ units in the second hidden layer [8]. Figure 9 is the network topology to implement Kolmogorov's theorem.

But this theorem only guarantees the existence of such a three-layer network and there is no known constructive method for finding particular activation functions required for such a network. In other words, there exists a particular three-layer

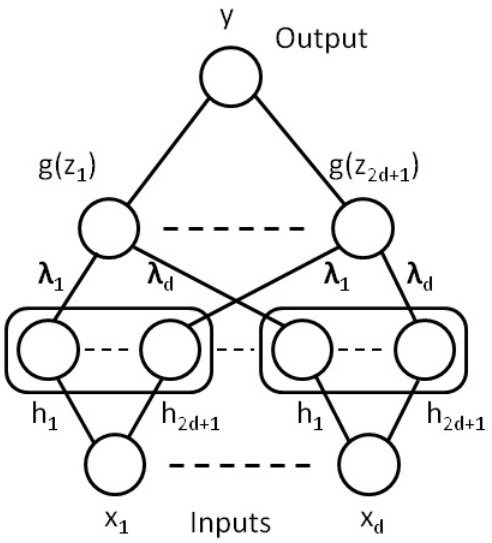


Figure 9: Network topology to implement Kolmogorov's theorem [8]

ANN which works as a universal approximator but not every three-layer ANN can be trained to be a universal approximator by just adjusting its connection weights. Moreover, just changing the number of hidden units doesn't guarantee to reach that particular approximator. To find such a three-layer universal approximator we may have to abandon the most fundamental characteristics of the ANN, i.e., the ANN as a collection of *simple* computational units because the universal approximator requires non-trivial combination of particular activation functions which are known to depend on the output function itself. Therefore, as shown by Kurkova et al., the theorem's relevance to practical ANN application is at best limited [8]. By interpreting Kolmogorov's theorem arbitrarily, assuming a simple MLP architecture as a potential universal approximator yields a large number of *false-negative* results in the ANN application literature [26].

Cybenko's universal, single-hidden layer network theory had also proven the very existence of such a network with the speculation on its practical applicability as follows [19]; *"While the approximating properties we have described are quite powerful, we have focused only on existence. The important questions that remain to be*

answered deal with feasibility, namely how many terms in the summation (or equivalently, how many neural nodes) are required to yield an approximation to a given quality? What properties of the function being approximated play a role in determining the number of terms? At this point, we can only say that we suspect quite strongly that the overwhelming majority of approximation problems will require astronomical numbers of terms. This feeling is based on the curse of dimensionality that plagues multidimensional approximation theory and statistics."

Therefore, the efforts to search an appropriate and, hence, efficient network connection structure is practically meaningful activity which does not contradict to the existence theorems for the universal approximator. In their research article on the neuroevolutionary method, NEAT (Neuro-Evolution by Augmenting Topology), Stanley and Miikkulainen dealt with this issue like this [80]; *"The basic question, however, remains: Can evolving topologies along with weights provide an advantage over evolving weights on a fixed-topology? A fully connected network can in principle approximate any continuous function (Cybenko, 1989). So why waste valuable effort permuting over different topologies? ... This article aims to demonstrate the opposite conclusion: if done right, evolving structure along with connection weights can significantly enhance the performance ..."* Without losing generality of their intention, the term *'evolving'* can be replaced by other term such as *'learning.'*

2.2.3 Feed-forward Network Architectures

Among all the possible network topologies, constraining internal connections to feed-forward connections only resulted in so-called feed-forward ANNs. The argument in the current thesis does not necessarily need to be confined to the feed-forward networks but the fundamental algorithms provide more clear insight when we deal with this type of networks and, hence, from this point, the current thesis deals only with the feed-forward networks only.

The MLP architecture represents only part of possible feed-forward networks because it allows connections between two adjacent layers only. The more general feed-forward architecture allowing any feed-forward connection is called Generalized Multilayer Perceptron (GMLP) or Bridged Multilayer Perceptron (BMLP). Thus, the additional constraint for the MLP structure is the absence of the cross-layer connections. By employing the cross-layer connections such as in GMLP and BMLP, the networks are more transparent for signal propagation, and they are easier to train [90]. In typical MLP networks, both the forward signal and backward error propagations must pass more nonlinear elements or computational units than in BMLP networks [90].

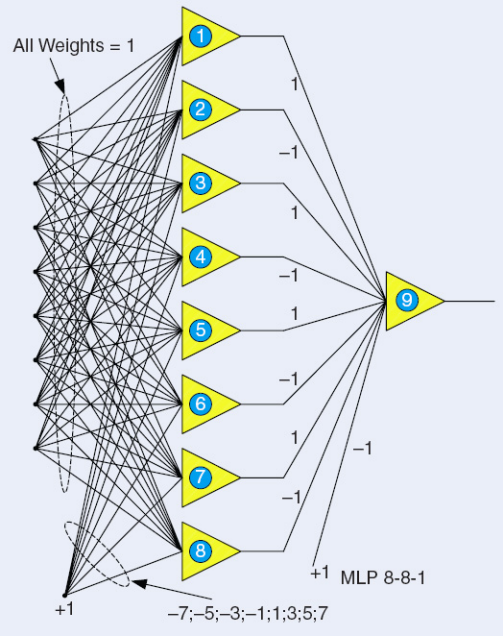


Figure 10: Bipolar neural network for the parity-8 problem with single-hidden layer MLP structure [90]

At the extreme case of establishing all possible feed-forward connections, the network structure becomes the Fully-Connected Cascade (FCC) which is also belong to GMLP or BMLP class of network. In this case, the network design problem has only one degree of freedom, i.e., the number of the hidden units. For the given number of hidden units, the FCC is the potentially most powerful structure due to its maximum

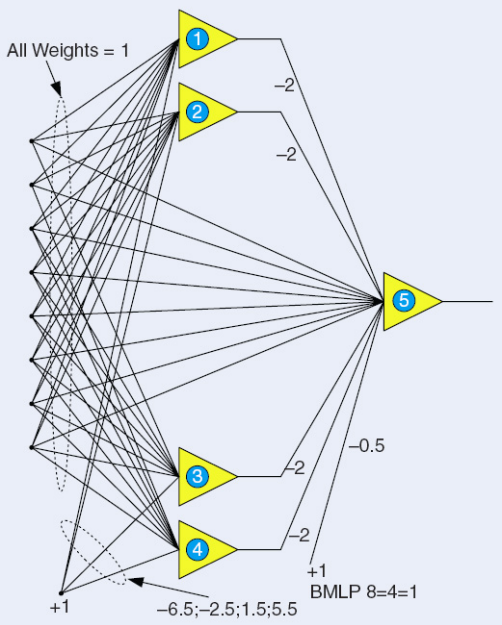


Figure 11: Bipolar neural network for the parity-8 problem with BMLP structure [90]

connectivity while the computational resources required for the training procedure also escalate to the maximum.

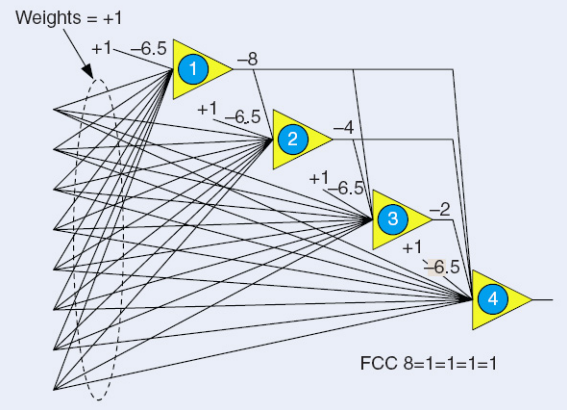


Figure 12: Bipolar neural network for the parity-8 problem with FCC structure [90]

Wilamowski calculated the minimum required numbers of the computational units and connections for the N-parity problems adjusting network architectures [90]. Detailed logic for the determination of those numbers are provided by Wilamowski [91].

Figure 13 indicates that different connection structures require dissimilar numbers

TABLE 1–NUMBER OF NEURONS/WEIGHTS REQUIRED FOR DIFFERENT PARITY PROBLEMS USING NEURAL NETWORK ARCHITECTURES.

ARCHITECTURE	PARITY-3	PARITY-7	PARITY-15	PARITY-31	PARITY-63
MLP	**4**/16	**8**/64	**16**/256	**32**/1024	**64**/4096
BMLP	**3**/14	**5**/44	**9**/152	**17**/560	**33**/2144
FCC	**2**/9	**3**/27	**4**/70	**5**/170	**6**/399

Figure 13: Number of neurons/weights required for different parity problems [90]

of neurons and connections and, therefore, the computational resources for the training of those networks also have wide variety. The different level of the computational resource is also more evident as the size of the problem grows.

2.3 Fundamental Algorithms

An improvement of the ANN training efficiency might be achieved to develop a radically new training methodology but, before that event happens, the error back propagation and its integration into the numerically efficient second-order optimization will be the most fundamental cornerstones for the practical ANN applications, especially, for the surrogate modeling tasks. At present, they are also serving as the common ground from which many newer methods have been evolved. In this section, the elementary derivation of the error back-propagation algorithm and its integration into the Levenberg-Marquardt optimization scheme are briefly reviewed.

2.3.1 Error Back-Propagation Algorithm

The Back-Propagation (BP) algorithm as a gradient calculation logic for the Multi-Layer network has originated from Rumelhart, Hinton and Williams' work, *"Learning representations by back-propagating errors"*, published in the journal *'Nature'* in 1986 [74]. In essence, the BP algorithm consists of successive executions of the chain rule in the general calculus which requires the continuity and the differentiability in the forward signal and backward error propagations. The indispensible constituent of the BP algorithm is the sigmoid-type activation function which satisfies these conditions with nonlinear activation capability.

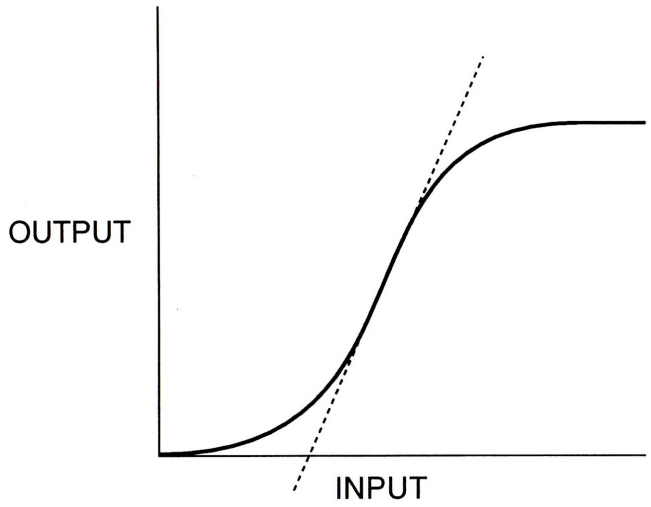


Figure 14: Differentiable, nonlinear and *"harmless-looking"* curve [17]

Training error can be defined as

$$E \equiv \sum_{n=1}^{N} E^n(y_1, y_2, \ldots, y_c) \tag{1}$$

where n is the index of the particular training example among the total N training examples, y is the network output, and c is the total number of output neurons. The derivative of error with respect to the weight w_{ji}, which for the connection from neuron-i to neuron-j, can be expressed as

$$\frac{\partial E^n}{\partial w_{ji}} \equiv \frac{\partial E^n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j \frac{\partial \sum_i w_{ji} g(a_i)}{\partial w_{ji}} = \delta_j g(a_i) \equiv \delta_j z_i \tag{2}$$

Here, g is the activation function and a_i is the output value of the neuron-i before activation by the activation function g. Because z_i is the simply activated output, $g(a_i)$, the only further required calculation is for δ_j which represents $\frac{\partial E^n}{\partial a_j}$. This δ has to be calculated by different manners for the cases of output neurons and hidden neurons. For output neurons,

$$\delta_k \equiv \frac{\partial E^n}{\partial a_k} = \frac{\partial E^n}{\partial y_k}\frac{\partial y_k}{\partial a_k} = \frac{\partial E^n}{\partial y_k}\frac{\partial g(a_k)}{\partial a_k} = \frac{\partial E^n}{\partial y_k} g'(a_k) \tag{3}$$

And for hidden neurons,

$$\delta_j \equiv \frac{\partial E^n}{\partial a_j} = \sum_k \frac{\partial E^n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial \sum_j w_{kj} g(a_j)}{\partial a_j} = g'(a_j)\sum_k \delta_k w_{kj} \tag{4}$$

Applying these equations, the residual error of each and every output node is propagated *back* to any arbitrary connection, w_{ji}, resulting in the value of δ_j and the input signal is propagated *forward* to that connection giving the value of z_i. Then, the wanted derivative of error with respect to that weight, w_{ji}, is obtained by simply multiplying these two terms, $\frac{\partial E^n}{\partial w_{ji}} = z_i \delta_j$.

In the above formulation, the bias term has, implicitly, been regarded as one of the connection weight whose input value has 1.0 value and not explicitly been expressed. Depending on the network implementation method, the bias terms can be attached to each and every neuron as the property of each and every computational unit or separate computation units whose output value is set to 1.0.

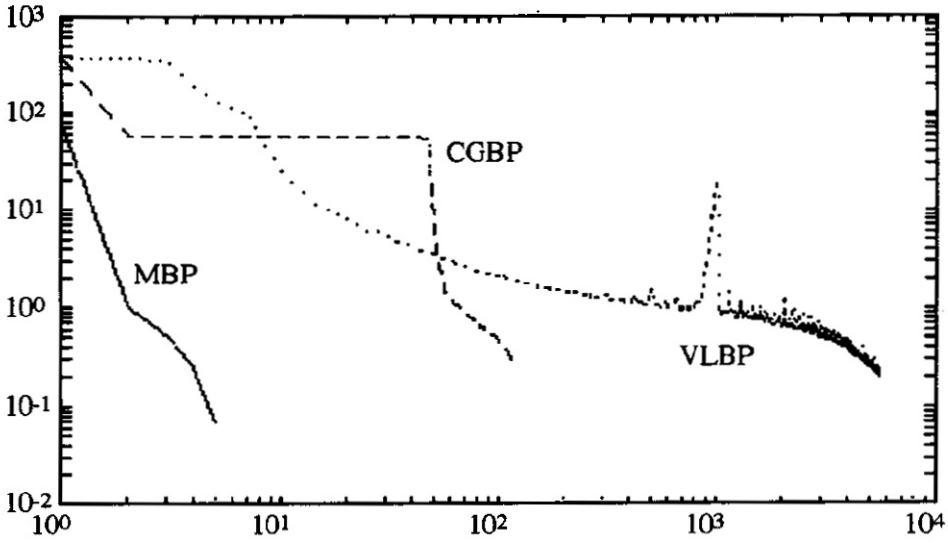
Although the BP algorithm is the backbone for the training of the MLP networks, as shown above, the BP algorithm does not necessarily limited to the layered structure in its mathematical derivation. As long as the feed-back connections are absent, the BP can operate on any feed-forward connection structure because entire formulation is centered on the individual connection weight. By preventing feed-back connections using such as assigning ascending indices for neurons and enforcing forward (from lower to higher index) and backward (from higher to lower index) propagation of the information, we can obtain exact error gradient for each and every connection weight. In this context, the BP has generality for the arbitrary feed-forward networks.

2.3.2 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm was formulated in 1944 by Levenberg and 1963 by Marquardt as an instance of a *model trust region* approach where the model is trusted only within some region around the current search point [8]. This approximation around the current search point is executed by using approximated Hessian matrix for the derivatives of error with respect to the weight neglecting cross-derivative terms. In 1994, Hagan and Menhaj published *"Training Feed-forward*

Networks with the Marquardt Algorithm" exploiting two advantageous characteristics of this algorithm for the training of the MLP networks. First, the LM algorithm originally designed for minimizing a sum-of-squares error, among other possible forms of error functions, which is the primary objective of the supervised learning. Second, the approximated Hessian naturally fit the information efficiently provided by the BP algorithm.

In this way, the LM algorithm takes advantage of the accuracy of the (pseudo) second-order methods and the numerical efficiency just explicitly using gradient information only.



NUMBER OF FLOPS REQUIRED FOR CONVERGENCE

	VLBP	CGBP	MBP
Sine Wave	8.42×10^7	1.75×10^7	1.89×10^6
Square Wave	2.28×10^9	1.49×10^8	3.48×10^7
2-D Sinc	2.94×10^9	7.67×10^8	1.07×10^8
4-D Test	—	7.71×10^9	1.97×10^9

Figure 15: Network convergence comparison (Sum of squares error versus epoch) [34]

Figure 15 is a part of result from Hagan and Menhaj's paper [34]. The upper chart shows the convergence history for their first test problem of the sine wave modeling

using a 1-15-1 network (one input node-15 hidden nodes-1 output node). Here, MBP is for the LM-BP algorithm, CGBP is for the conjugate gradient back-propagation algorithm, and VLBP is for the variable learning rate back-propagation algorithm. The CGBP is also one of the widely used second-order training algorithm which is based on the Fletcher-Reeves version in this work. The epoch on the abscissa means the number of the BP sweeps; one epoch corresponds to the one learning cycle for the entire training examples. This chart indicates the several orders of reduction in the required training epochs is realized by the BP-LM method from the other methods. The lower table in the same figure also shows that not only the number of training epochs but also the number of floating point operations reduces in case of using the BP-LM method.

The error to be minimized through the BP-LM method is redefined as

$$\bar{E} = \frac{1}{2}\sum_{n}(\epsilon^n)^2 = \frac{1}{2}\left\|\epsilon\right\|^2 \tag{5}$$

where ϵ^n is the error for n-th training data, and ϵ is a vector with element ϵ^n. Suppose we already have weight vector $\mathbf{W_{old}}$ and want to move for the improved new weight vector $\mathbf{W_{new}}$. If the difference is small, the linearized new error vector is

$$\epsilon(\mathbf{W_{new}}) = \epsilon(\mathbf{W_{old}}) + \mathbf{Z}(\mathbf{W_{new}} - \mathbf{W_{old}}) \tag{6}$$

where the matrix $\mathbf{Z}$ is defined with the element of

$$(\mathbf{Z})_{ni} \equiv \frac{\partial \epsilon^n}{\partial W_i}. \tag{7}$$

Here, without the loss of generality, notation for weight has been changed in which i for W_i means just i-th weight within the total free weight parameter set regardless of the corresponding connections. Later, we can assign this weight back to the correct connection. The term $\frac{\partial \epsilon^n}{\partial W_i}$ is the gradient of error function to the corresponding connection weight calculated by BP algorithm. But, in this case, the particular error

function corresponds to the pure training error which is the difference between the output value of the output neuron and the target value,

$$E = y_k - t_k \tag{8}$$

where t is the target output value. Hence the partial derivative of this error function to the output value of the output neuron becomes 1 making previous equation for δ_k for output neurons simply,

$$\delta_k \equiv \frac{\partial E^n}{\partial y_k} g'(a_k) = \frac{\partial (y_k - t_k)}{\partial y_k} g'(a_k) = g'(a_k) \tag{9}$$

This is the only required modification for BP algorithm to obtain sensitivity matrix $\mathbf{Z}$ in LM method [34]. Rewriting the error function for LM with linearized error vector,

$$\bar{E} = \frac{1}{2} \left\| \epsilon(\mathbf{W_{old}}) + \mathbf{Z}(\mathbf{W_{new}} - \mathbf{W_{old}}) \right\|^2 \tag{10}$$

The minimization of this error results in

$$\mathbf{W_{new}} = \mathbf{W_{old}} - (\mathbf{Z^T Z})^{-1} \mathbf{Z^T} \epsilon(\mathbf{W_{old}}). \tag{11}$$

For the previously defined sum-of-squares error function, the elements of the Hessian matrix take the form [9],

$$(\mathbf{H})_{ik} = \frac{\partial^2 \bar{E}}{\partial W_i \partial W_k} = \sum_n \left\{ \frac{\partial \epsilon^n}{\partial W_i} \frac{\partial \epsilon^n}{\partial W_k} + \epsilon^n \frac{\partial^2 \epsilon^n}{\partial W_i \partial W_k} \right\}. \tag{12}$$

If we neglect the second term, then the Hessian can be written in the form [9]

$$\mathbf{H} = \mathbf{Z^T Z}. \tag{13}$$

This is only exact in the linear interval of the weight space, and the numerical procedure always has to be ensured within the validity of this linearization. The LM method achieves this process by introducing λ parameter such as

$$\tilde{E} = \frac{1}{2} \left\| \epsilon(\mathbf{W_{old}}) + \mathbf{Z}(\mathbf{W_{new}} - \mathbf{W_{old}}) \right\|^2 + \lambda \left\| \mathbf{W_{new}} - \mathbf{W_{old}} \right\|^2 \tag{14}$$

where the parameter λ governs the step size. The minimization of this approximated error results in

$$\mathbf{W_{new}} = \mathbf{W_{old}} - (\mathbf{Z^T Z} + \lambda \mathbf{I})^{-1} \mathbf{Z^T} \epsilon(\mathbf{W_{old}}) \tag{15}$$

where $\mathbf{I}$ is the identity matrix.

2.4 Literature Review on Network Development Methods

Designing a task-specific neural network is one of the most pursued but still challenging problems in the field of the artificial neural network [11]. Compared to the maturity and the numerical efficiency of the weight optimization methods, no well developed theory has definitely been found yet to determine optimal network size (i.e., the numbers of computational units and connections wiring them) and appropriate wiring structure other than task-independent, regular structures such as Multilayer Perceptron (MLP) [37]. Therefore, while the theoretical definition of *learning* covers all of the network properties including the way in which the neurons are connected, the activation functions, and even the learning rules themselves, the current paradigm of neural network training is confined to *learning the weights* leaving the other determinations in the network design tasks to the judicious selections by human designers [63].

In theory, Cybenko's work on the *universal appoximator* reduced the vast amount of degree-of-freedom in network design to a single parameter, the number of hidden units in the single-hidden-layer MLP, which was shown to be able to approximate arbitrary decision regions as long as the enough number of hidden units are provided [19]. But, in practice, the required number of hidden units can increase to be *"astronomical"* as speculated by Cybenko himself [19] and witnessed in many real-world cases in which MLP networks frequently suffer scalability problem as the dimensionality of the task and, hence, the size of network grows [41]. Obviously, this problem seems to be beyond the remedy of simply adjusting the number of hidden units.

Therefore, as though Cybenko's work provides an important mathematical implication for the sensitivity analyses to determine a proper number of hidden units in MLP, the architectural design problem of neural network which is not limited to the MLP architecture is still relevant and becomes a paramount issue, especially, when the computational feasibility or efficiency is critical in network training process.

Moreover, the architecture of neural network does not only govern the computational cost during training phase but also significantly affects the generalization performance in prediction phase in which network is required to estimate reasonable outputs for unseen inputs existing outside the set of training examples [35, 53, 43]. Generally speaking, for the given set of training examples, too small network has fundamental limitation on its learning capability and too large network results in poor generalization performance characterized by phenomenon such as over-fitting [51, 70].

The impact of overall network size on generalization performance has driven the needs for development of the constructive methods [24, 29, 51] and the pruning methods [52, 36, 70]. The constructive methods such as Dynamic Node Creation (DNC, [3]), Cascade-Correlation (CC, [24]), and Projection-Pursuit (PP, [44]) begin training with a small (or minimally sized) network and introduce new network components such as new connections or new computational units whenever the network converges to local optimum until the satisfactorily working, minimum-size network is obtained. Among the various pruning methods, the penalty-term methods share the conventional weight optimization procedure except modification in the objective function which has the additional term penalizing larger weights in their magnitude, promoting less contributing connection weights to decrease their magnitudes, which will be pruned by application of the numerical threshold. The sensitivity-based pruning methods such as Optimal Brain Damage (OBD, [52]) and Optimal Brain Surgeon (OBS, [36]) begin training with the post-training networks, estimate the sensitivity (to the training error) of each network component around its elimination, and the

components having the less-than-threshold sensitivities are eliminated before the final retraining. Confronted with scalability problem in network training, the growing or constructive methods have advantage, in general, over the pruning methods requiring larger-than-necessary, initial network because they search for proper size of the networks starting from the smaller networks. Assuming that there might exist more efficient internal connection structure among the networks with similar size (i.e., in number of computational units and connections), these growing and pruning methods have only limited capability to exploit the potential benefits of rewiring network structure because these methods operate on pre-defined, particular types of network architecture such as strictly layered MLP (in case of DNC and PP) or Fully-Connected Cascade (FCC [90], in case of CC).

Differently with these hill-climbing network sizing methods, the efficacy of stochastic topological exploration is intensively being discussed in the field of neuroevolution in which the evolutionary algorithms such as Genetic Algorithm (GA) or Genetic Programming (GP) are utilized to evolve task-specific neural networks by directly adjusting the broader set of network properties such as the number of neurons, the wiring structure, and the types of activation functions [48, 32, 4, 93, 62, 80, 27]. The evolutionary approach can produce, topologically, more diverse network configurations than the hill-climbing methods, finding useful network structures but this strategy accompanies computationally expensive, population-based heuristic search processes. For real-world problems for which even a single training campaign costs significant amount of computational resources, the accumulation of total training cost for the multiple generations each of which consists of multiple individual networks as a population group can easily become computationally prohibitive. This is one reason for the synthesis of the artificial evolution and the connectionist learning is currently applied more actively to relatively small-sized networks.

In view of the mechanism by which network is adjusted (or trained) towards reduced training error, both the constructive methods and the evolutionary approach use two, hierarchically decoupled search processes; an *inner* process for weight optimization and an *outer* process for structural enhancement. In the evolutionary approach, the outer process corresponds to the artificial evolution consisting of repeated cycles of selection, reproduction, and mutation operations for each individual network and the constructive methods also repeat inner process of weight optimization whenever discrete introduction of new network components occurs. In other words, there exists a prevailing dichotomy between weights and connectivity in the neural network training. Letting the weight optimization process intact, the connectivity of network has to be determined by empiricism, or adjusted only in its scale following the stereotyped architecture, or evolved. The main hypothesis of this paper is that, without breaking this dichotomy between weights and connectivity in training process, an efficient design of a task-specific neural network is very hard to achieve because, as long as connectivity and weights are searched by separate means, a structural improvement of the neural network requires either repetitive re-training procedures or computationally demanding topological meta-search cycles. The idea of infusing connectivity learning capability into the well-established weight optimization algorithm such as the Error Back-Propagation (EBP) algorithm has attracted relatively limited attention as a network design strategy. As one example of research in this direction, KrishnaKumar reported a connectivity adjusting learning method introducing *connectivity parameter*, which is harnessed by logistic sigmoid function or *connectivity function*, as a multiplication factor to each and every connection weight. The modified EBP learning process does not only optimize weights but also connectivity parameters, whose value determine the deletion (if connectivity function results in 0.0) or survival (otherwise) of the corresponding connections [49]. This method, by optimizing connectivity as well as weights, produced much sparser networks (in their

number of connections) which outperformed MLP networks (which have the same number of neurons) trained by conventional EBP algorithm in the supervised learning task of mapping the aero-mechanical input data to the helicopter performance data [49].

2.4.1 Neuro-Evolution Approach

Since early 90's, the combination of the ANN and the evolutionary algorithms such as Genetic Algorithm (GA) and Genetic Programming (GP) became one promising research area with the aim of automatic development of the ANN. Among the diverse approach, the main strategy is to optimize both the network structure and the weight parameters in a single framework of the evolutionary algorithm, which resulted in the methods belong to the TWEANN (Topology and Weight Evolving Artificial Neural Network) strategy. Evolving network structures without any weight information makes it harder to access the real performance of a given topology due to the noisy fitness evaluation problem [93]. In other words, for the same structure, random weights can result in the dissimilar performance and, therefore, the TWEANN approach is more reasonable than topology-alone search strategies.

Obviously, this approach has been inspired partly from the biological adaptation concepts of the evolution and the learning as described by Belew et al. [4]; *"It is extremely appealing to consider hybrids of neural-network-based learning algorithms with evolutionary search procedures, simple because Nature has so successfully done so."*

Besides the inspiring biological plausibility, the popularity of this approach has originated from the fact that the evolutionary, population-based, heuristic search has been believed as one of the very few feasible methods to explore vast design space of network topology. Floreano et al. [27] expressed that the artificial evolution is extremely efficient, or even unique, when it is used to explore both the topology and

the adaptive properties of the ANN.

As reviewed in the survey article by Yao [93], diverse methods combining evolutionary algorithm and the ANN have been introduced and widely applied in the fields of supervised, unsupervised, and reinforcement learning such as for robotics and system control applications. Due to the unique nature of the evolutionary algorithms based on the discrimination between the genotype, i.e., the encoded chromosome string for the network topology in this case, and the phenotype, i.e., the realized and fully developed network entity, the efficient representation of detailed network properties into a compact chromosome string has been a single most important issue.

Currently, more than 300 papers can readily be identified in the literature related to the genetic encoding schemes for ANNs [77]. But the proper and right way of encoding ANN structure into compact form which enables network topology and weight optimization is still being sought. In 2007, Benardos and Vosniakos described the current status of the TWEANN methods as this [5]; *"Increased attention is especially directed to proposing a systematic way to establish an appropriate architecture in contrast to the current common practice that calls for a repetitive trial and error process, which is time-consuming and produces uncertain results. Despite the increased level of research activity, the described problem has not yet been answered definitely."*

In the context of the surrogate modeling, the TWEANN methods are computationally too expensive to be applied for real world problems while they are successfully applied in smaller-sized problems. Majority of the surrogate modeling tasks such as for CFD analysis results requires significant computational resources for a single campaign of training just using a few candidate network topologies. Considering the required number of multitudes of such trainings in case of evolutionary process which consists of multiple generations of populations, the TWEANN approach is practically infeasible. Therefore we have to avoid the adoption of the TWEANN methods and to focus on more computationally light methods.

But the combination of two most fundamental forms of adaptation such as evolution and learning is an ultimate goal of the neural network design as long as the practical mean to it having been devised and matured enough. Implementation of the massive parallelism which is intrinsic to the evolutionary algorithms and to the calculation mechanism of the ANN might be one direction to be investigated further to realize this goal. Moreover, the way the TWEANN methods evolve topology and learn the weights has a great chance to be improved in case of using unconventional learning paradigm which is described in the next chapter.

2.4.2 Constructive Methods

The general idea of the constructive methods is to start with a small network, then add hidden units and connections incrementally until a satisfactory solution is obtained. This approach is conceptually opposite to the pruning methods, which are described in the subsequent section, starting from the superfluously large network and eliminating less contributing or unnecessary connections and hidden units. Following Kwok et al. the advantages of the constructive methods over the pruning methods are [51];

1. It is straightforward to specify an initial network, whereas for pruning methods, it is more difficult to determine how big the initial network should be.

2. Constructive methods are computationally more economical than pruning methods by starting from a small network.

3. Assuming multiplicity of satisfactory network architecture, constructive methods have greater chance to find a minimal network among them, also by searching from small networks and, then, larger networks later.

An important issue relevant to every constructive method is when to stop the network growth. Stopping criteria depending only on the training error does not guarantee the satisfactory generalization performance because of the bias-variance

tradeoff. Letting f as a general relationship between input and output spaces, the generalization performance of network $f_{n,N}$ defined by n hidden units and trained by N training patterns can be expressed as the function of the distance between f and $f_{n,N}$ [51].

$$E(\|f - f_{n,N}\|) \tag{16}$$

This error is known to come from two sources, the approximation error (bias, $\|f - f_n\|^2$) and the estimation error (variance,$E(\|f_n - f_{n,N}\|^2)$). Barron [51] showed that, for the networks with a single hidden layer using sigmoid activation function, above error is bounded by

$$O\left(\frac{c_f^2}{n}\right) + O\left(\frac{nd}{N} log N\right) \tag{17}$$

where d is the input dimension of the network and C_f is the first absolute moment of the Fourier magnitude distribution of f. The first term comes from the approximation error and the second term from the estimation error. Thus, when the number of hidden units, n, increased, approximation error decreases but the estimation error increases. Therefore, it is important to determine stopping criteria for network growth for good generalization performance. Widely used formal methods are cross-validation, Alkaike's information criteria (AIC), and Bayesian information criteria (BIC), and minimum description length (MDL), etc. Practical substitution to these formal methods is terminating training when the training error is below a certain threshold or validation error reaches the minimum value.

2.4.2.1 Dynamic Node Creation

The representative work in this category of constructive methods was originally proposed by Ash [51]. As training proceeds, sigmoid hidden units are added one by one to the same hidden layer. The entire network is retrained completely whenever hidden unit is added. This is a faithful implementation adopting the single-hidden-layer universal approximator theory by Cybenko [19] and, theoretically, this method can reach

that universal approximator. However, as Cybenko himself mentioned, the practical scale problem might require *'astronomically many'* hidden units and the required repetitive re-trainings significantly limit the applicability of this type of methods.

2.4.2.2 Cascade Correlation

Cascade Correlation (CC) had been formulated by Fahlman and Lebiere [24] and became one of the popular supervised learning architecture that dynamically grows fully-connected layers. CC begins with a minimal network, then automatically trains and adds new hidden units one by one.

The CC operates on the one particular network architecture, the fully-connected cascade. This architecture connects every possible feed-forward connections between computational units having sigmoid activation functions whereas the MLP structure assumes internal layout determined by the number of hidden layers and the number of hidden units in each hidden layer not allowing cross-layer connections. Thus the structure of network in the CC is completely determined by the number of input and output nodes and the number of hidden layers which has only one hidden unit in each layer.

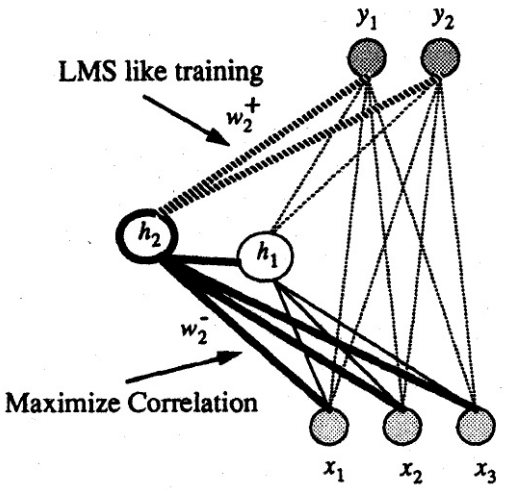


Figure 16: The structure of a Cascade Correlation learning network [44]

When a new hidden unit is added, it is added as the last hidden layer or node

between the previously existing last hidden unit and the output nodes. The input connections, incoming from all the other nodes except output nodes, to the newly added hidden node are assigned with the weight values calculated from the maximization process for the correlation between the activation value of the newly added hidden unit and the residual error in the output nodes.

At this point, the most distinctive feature of the CC to the conventional learning methods is that, including the newly assigned input weights, all the existing connection weights are frozen except the weights from that newly added node towards output units. Here *'frozen'* means that they are regarded as fixed constants which are excluded from the remaining weight training procedure.

And the method trains the flexible weights and the above process is repeated until satisfactory training result is obtained. Figure 16 shows the network structure of CC where the hidden node h_2 has just been added to the network.

By starting with a single hidden node network, this method results in automatically sized fully-connected cascade architecture. Reduction of the number of adjustable weight parameters greatly accelerates training procedure exploiting second-order Newtonian method-based weight optimizer called QuickProp [24].

2.4.2.3 Projection Pursuit

The CC adds a new hidden unit to approximate remaining residual error which could not be captured by existing computational units. A Projection Pursuit (PP) network accomplishes the same objective of approximating complex mapping of residual error by adding a hidden unit having *'trainable'* activation function within a single-hidden layer architecture without cascade connections [44]. The PP learning network is a statistical procedure for multivariate data analysis using a single-hidden, layer feed-forward network architecture where the hidden layer has a special type of activation function and the output layer has linear nodes (Figure 17). This scheme aims to

interpret high-dimensional data through well-chosen lower-dimensional projections. The *"pursuit"* refers to optimization according to the direction of projection [44].

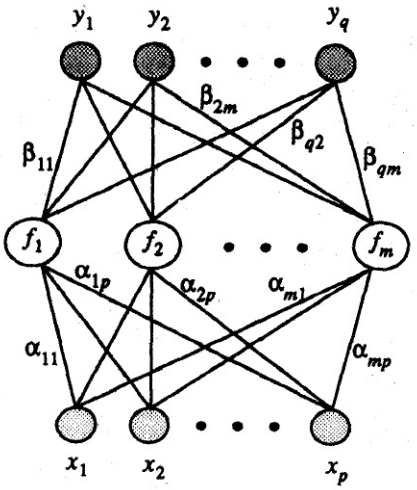


Figure 17: The structure of a Projection Pursuit learning network [44]

Similar to the MLP training, the PP forms the projections of the data in directions defined by the network connection weights but the PP optimizes the activation functions themselves involving one dimensional data-smoother or approximator while the MLP employs a set of fixed nonlinear activation functions such as sigmoid functions in hidden units.

The usual training procedure for the PP network is, first of all, to optimize the parameters related to the newly added hidden unit including the coefficients for the activation function and the connecting weights to the previously existing nodes and, then, to perform *back-fitting* to fine tune the parameters associated with the previously existing hidden nodes.

In this way, starting from the minimal single-hidden layer network, dynamically growing learning network can be built by optimizing connection weights as well as the functional coefficients for the activation functions.

2.4.3 Pruning Methods

Over-fitting is regarded as happening when the network has more degrees of freedom (the number of connection weights and biases, approximately) than those of the underlying functional relationship between inputs and outputs of training data [70]. In general, a rule of thumb to avoid over-fitting is to use the smallest possible model that will fit the data accurately enough. In case of having the limited degrees of freedom, the model will use them to adapt to the most distinctive regularities existing in the data and will be less dependent on the noise. This is the aim of network pruning techniques with subsidiary benefit of the small network such as being faster to predict, light to construct, and easier to understand.

The pruning method is implemented either as off-line or on-line strategy [44]. The off-line strategy prunes the "redundant" or "insignificant" connections and/or hidden units "after" a large size network is trained to improve the generalization capability, where the estimation of the sensitivity information, i.e., the impact of pruning for connection weights and/or hidden units, is important. The Optimal Brain Damage (OBD) and the Optimal Brain Surgeon (OBS) methods belong to this class of techniques. The off-line strategy prunes the connections and/or hidden units during the training process by adding in the network fitness evaluation function a regularization term which enforces the weights of small magnitudes to converge towards zero value.

2.4.3.1 Sensitivity Methods

Sensitivity methods estimate the impact of the pruning of connections and/or hidden neurons. Because elimination of existing connections usually deteriorate training error while the generalization error is possibly decreased, this type of methods use the estimation of sensitivity to minimize the deterioration of training error. One of the typical approach has been formulated by Karnin [70] as follows; A sensitivity of

a particular connection weight w is given as

$$S = -\frac{E(w^f) - E(0)}{w^f - 0} w^f \tag{18}$$

where w^f means the final value of the weight after training and 0 is its value upon removal, and $E(0)$ is the error when it is removed. The objective is to remove the connection weight which has the allowably small value of the sensitivity. Instead of a brute force evaluation for all existing connections after the training finished, the sensitivity S can be kept on track of the whole training procedure, approximately,

$$\hat{S} = -\sum_{n=0}^{N-1} \frac{\partial E}{\partial w} \Delta w(n) \frac{w^f}{w^f - w^i} \tag{19}$$

where N is the number of training epochs and w^i is the initial weight. In case of using the back-propagation learning rule, this becomes

$$\hat{S} = -\sum_{n=0}^{N-1} [\Delta w(n)]^2 \frac{w^f}{\eta(w^f - w^i)}. \tag{20}$$

After training, each weight has an approximated sensitivity and the lowest sensitivity weights can be removed. An isolated neuron as a result of pruning for all its input connections becomes a constant output neuron whose outputs can be included all the connecting output neurons' biases. Then, the neuron can be removed from a network without making any difference. This is one of the typical methods using gradient information to estimate the impact of connection removal although there are several variations in the way the sensitivity is estimated.

2.4.3.2 Optimal Brain Damage

The Optimal Brain Damage (OBD) is also one of the sensitivity-based pruning methods but it measures the sensitivity using the approximated second derivative of the error with respect to the connection weight [52]. When the weight vector $\vec{W}$ is perturbed, the change in the training error is approximately

$$\delta E = \sum_i g_i \delta w_i + \frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta w_i \delta w_j + O(\|\delta \mathbf{W}\|^2) \tag{21}$$

where the δw_i's are the components of $\delta\vec{W}$, g_i are the components of the gradient of E with respect to $\vec{W}$, and the h_{ij} are elements of the Hessian metrix $\mathbf{H}$;

$$g_i = \frac{\partial E}{\partial w_i} \text{ and } h_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \tag{22}$$

Although there exist several methods to exactly compute the Hessian matrix [7], this task usually require enormous amount of computation ($6.5x10^6$ terms for a 2600 parameter network, [52].) The OBD assumes that the δE resulted by deleting several parameters is the sum of the δE's resulted by deleting each parameter individually. This "diagonal" approximation neglects all the cross-terms. Moreover, the OBD assumes "extremal" approximation, that the pruning will be executed after the training has converged, which means the gradient term can be neglected and any perturbation will make E to increase or stay the same. Moreover, applying the "quadratic" approximation, the higher-order error term is also neglected. These three approximations reduce the above equation to

$$\delta E = \frac{1}{2}\sum_i h_{ii}\delta w_i^2 \tag{23}$$

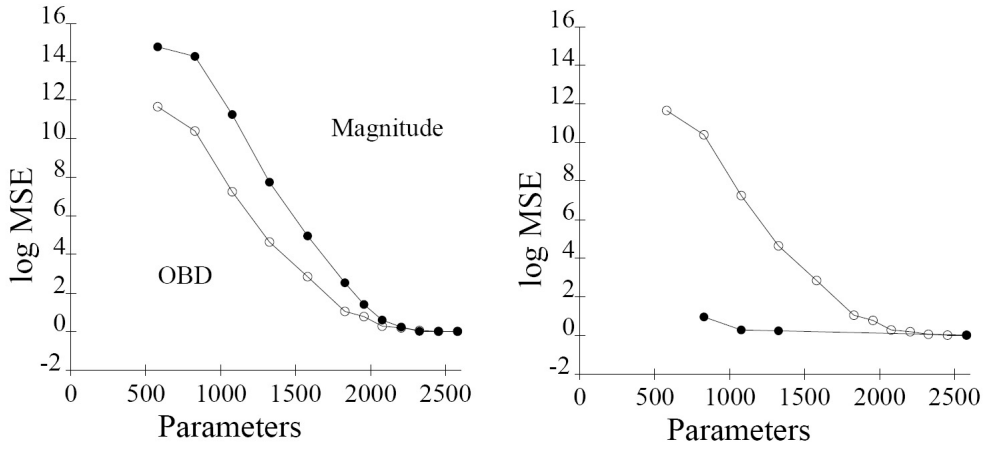


Figure 18: The effect of the Optimal Brain Damage (OBD) method [52]

Figure 18 shows the effect of the OBD method applied on the hand-written zip-code digit recognition problem. The initial network has $2,578$ free parameters over 10^5

connections and trained using 9300 examples and tested using 3350 examples. The left figure shows the deterioration of the training error as the number of pruned connections increases (from right to left on the abscissa) where the upper curve is the result of the magnitude-based pruning and the lower curve is the result of the sensitivity-based OBD method. The OBD method resulted in the less damaged training result compared to the magnitude-based pruning where simply the smallest weights are eliminated. The right figure is the result before and after re-training. Hence, the OBD with re-training can produce nearly identical modeling performance with almost half the number of connections.

Later, the Optimal Brain Surgeon (OBS) method eliminated the "diagonal" approximation allowing the more exact implementation of the Hessian matrix and showed the improved pruning performance [36].

2.4.3.3 Penalty Term Methods

The penalty term methods modify the objective function for training other than usual least-square of the residual error between target and predicted values so that normal error back-propagation learning rule effectively drives the weights to smaller values and, then prune the connections having the smaller weights than pre-defined threshold. While there are diverse form of the modified cost function, the following simple form represents the essence of this type of approach [70];

$$C = \sum_k (t_k - y_k)^2 + \lambda \sum_{i,j} |w_{ij}|. \tag{24}$$

where t_k and y_k are the target and the predicted values and λ is so-called regularization factor. The second term adds $-\lambda sgn(w_{ij})$ to the weight update rule. If $w_{ij} > 0$, the weight is decreased, otherwise, if $w_{ij} < 0$, then it is increased depending on the values of λ.

2.4.4 Applicability Issues

The evolutionary approach provides a valuable conceptual framework integrating the two most fundamental forms of adaptation, i.e., evolution and learning. But the current methods in this field almost unanimously are based on the strictly separated hierarchical optimization loops. The network topology is optimized by the higher level evolutionary manipulation and the connection weights are trained by individual weight optimization. Therefore, in many practical cases where a single training campaign consumes a considerable amount of computational resources, the escalated scale of the solution is usually required as much as the number of the generations and the size of the network populations. This is the reason for this type of methods to prosper in the relatively small-scale problems rather than in the surrogate modeling area.

The pruning methods result in the more parsimonious networks and, hence, are the valuable tools to reduce the network for the given task without a significant deterioration in the prediction performance. The basic idea behind the need for pruning is that larger-than-necessary networks, in general, have the lower generalization performance by causing over-fitting problem.

The advantage of the pruning approach is more valuable when we have already obtained the larger-than-necessary network. In the practical scale surrogate modeling tasks, the more relevant issue is how to find the way to obtain a performing network. In this sense, the majority of the network design or selection issues reside in the smaller-than-necessary side due to the limited computational resource rather than the larger-than-necessary side where the pruning methods start to act. Moreover, the benefit by reducing the network size by removing less-significant connection weights or computational nodes can be offset by the additional computational cost for re-training procedures. Therefore, in the context of surrogate modeling tasks, the constructive methods deserve more attention than the pruning methods.

The three constructive methods reviewed previously have limited applicability to the surrogate modeling tasks;

- *Dynamic Node Creation* operates by the train-add-retrain cycles on starting from the baseline MLP network. This means that a unit adjustment requires repetition of a full training process. For the practical scale problems requiring hundreds to thousands connections need more computationally efficient algorithm.

- *Cascade Correlation* constructs a Fully-Connected Cascade (FCC) network which is one extreme architecture belonging to the MLP network category. While the usual FCC architecture consumes the maximum amount of computational time and cost for its training due to its maximal feed-forward connectivity, the CC trains only the part of those connections with fixing the others trained before the addition of the new computation node, i.e., the last hidden unit. And the input weights connecting all the lower-layer nodes to the newly added hidden node are determined by the correlation maximization. The maximization of correlation between node ouput and the prediction error can be one reasonable choice to efficiently calculate the input weights to the new hidden units but this strategy is not always guaranteed to result in the optimal weights. In other words, there is possibility for the CC to result in the larger network than necessary because of the early exclusion of potentially useful optimization parameters and the successive addition of new nodes being the only way to improve the network. Moreover, the exclusion of weight parameters might be beneficial only when the remaining adjustable weight parameters can handle the variability of the training error. Hwang et al. pointed out the limited mapping capability and the unit saturation problem of the CC method resulting in degraded regression analysis performance [44]; *"it is shown that the maximum correlation*

training criterion used in a CC tends to produce hidden units that saturate and thus makes it more suitable for classification tasks instead of regression tasks as evidenced in the simulation results." Therefore, while the unique features of the CC define the method as the one of its kind, its applicability to the general Multi-Layer network which operates on the more sparsely connected topology is, at best, limited.

- *Projection Pursuit* can also grow a working network starting from the minimal structure. In this case, the network topology is based on the single-hidden layer network only and the characterizing feature distinctive from the conventional neural network training method is the adoption of the optimizable activation functions in the hidden units. Instead of using the Multi-Layer structure, the part of the ability of the MLP to hierarchically exploit the data feature detected by the previous layer can be thought to be transferred to the more versatile activation functions whose parameters are also optimized during the training process. But the inclusion of the functional form of the activation function as a adjustable parameter escalates the scale of the network design problem additional to the selection of the network topology and weight optimization because the performance of the PP depends on the judicious selection of the types of the optimizable activation functions [44]. Therefore, this strategy poses a significant dissimilarity for the general application to the Multi-Layer network models which are the most popular architecture for the surrogate modeling, especially, adopting the BP-LM training convention.

While the diverse constructive methods such as the Cascade Correlation and the Projection Pursuit are widely applied to the supervised learning tasks, the majority of the practical surrogate modeling tasks are executed by the Multi-Layer network architecture consisting of the computational units equipped with the sigmoid-type,

fixed, and, hence, simple activation functions. Since the advent of the error back-propagation algorithm, the BP has been the standard mechanism to propagate the effect of training error to the weight level and changed its role from the weight updating rule to the gradient calculator as the more efficient second-order optimization techniques such as the Levenberg-Marquardt algorithm being introduced to the ANN training tasks. In this context, the network construction methodology which is particularly suitable to the general Multi-Layer network architecture and its training framework is still being sought. A few of research works relevant in this aspect are briefly reviewed in the next section.

2.4.5 One Noteworthy Method

In the ANN literature, there exist more exotic methods which are not easily determined to belong to any of previously mentioned classes of network design methods. Some of them are particularly noteworthy because they possess potentially very suitable applicability to the Multi-Layer network architecture by directly optimizing connectivity itself during the training process without the need for repetition of training.

KrishnaKumar developed a connectivity adjusting learning method introducing additional sigmoid function as the property of each and every connection weight [49]. The basic idea is to represent the weight as the multiplication of the weight value w_{ji} and the connectivity function $g(C_{ji})$ which is the logistic sigmoid (Figure 19). At each training epoch, connectivity parameter, C_{ji}, as well as the weight, w_{ji}, are updated resulting in that some of the connections have whole weight but the remainders reduce to zero. In this formulation, the sum of the input signal for each neuron is

$$a_j = \sum_{i=1}^{j-1} w_{ji} g(C_{ji}) z_i \tag{25}$$

where

$$g(C_{ji}) = \frac{1}{1 + e^{-\alpha C_{ji}}}, \alpha > 10 \tag{26}$$

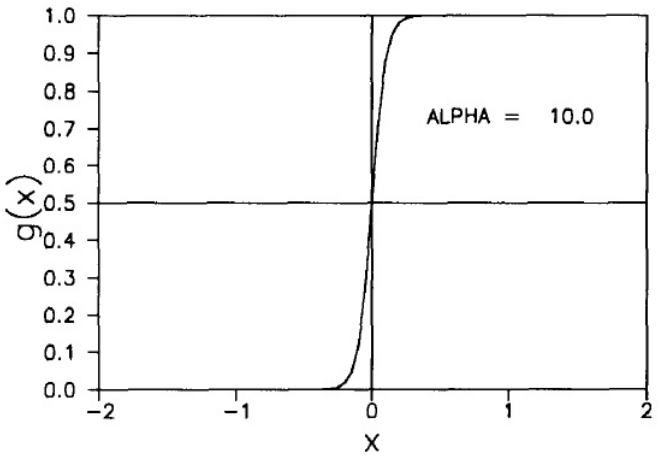


Figure 19: The connectivity function [49]

And the connectivity parameter is updated in the similar fashion with weight update in the BP learning rule;

$$w_{ji} = w_{ji} - \eta \frac{\partial E}{\partial w_{ji}} \tag{27}$$

$$C_{ji} = C_{ji} - \lambda \frac{\partial E}{\partial C_{ji}} \tag{28}$$

where η and λ are the learning coefficients.

The method has been applied to the model identification problem for the UH-1 helicopter [49]. Not only the number of the connection has been reduced compared to the double-hidden layer MLP but also the prediction performance has been improved. Figure 20, Figure 21 and Figure 22 are the network layouts, convergence histories, and the target versus prediction plots showing the comparison between the optimized GMLP and the baseline MLP networks.

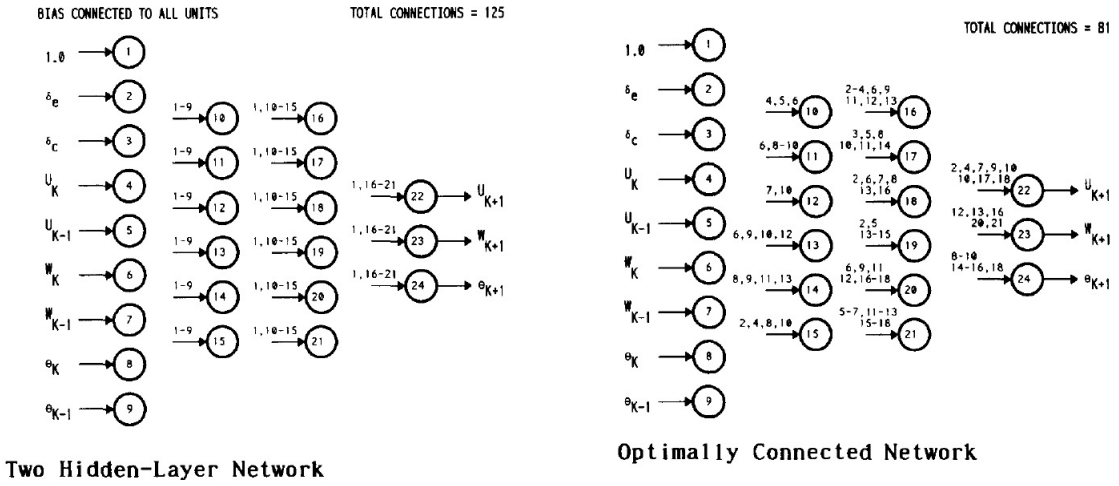


Figure 20: The network structures for the UH-1 helicopter model identification problem [49]

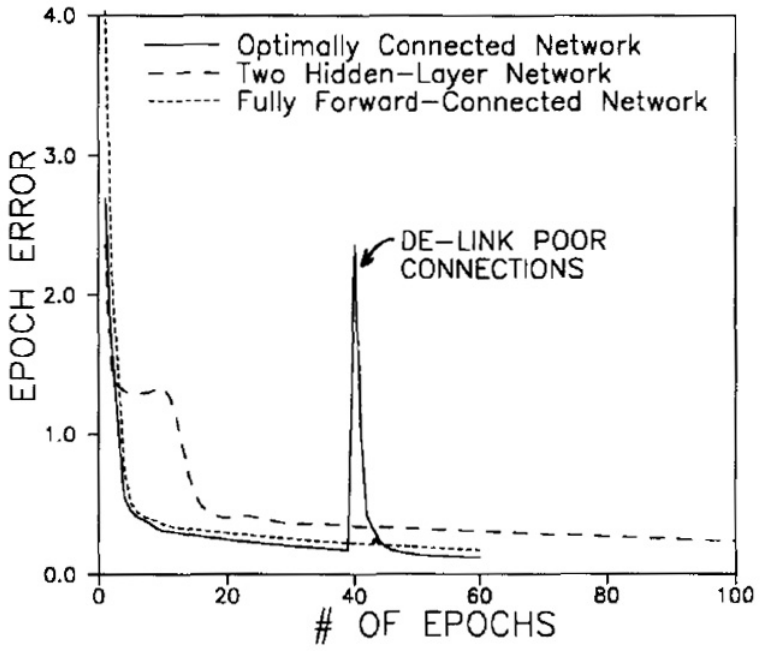


Figure 21: Error curves for the UH-1 helicopter model identification problem [49]

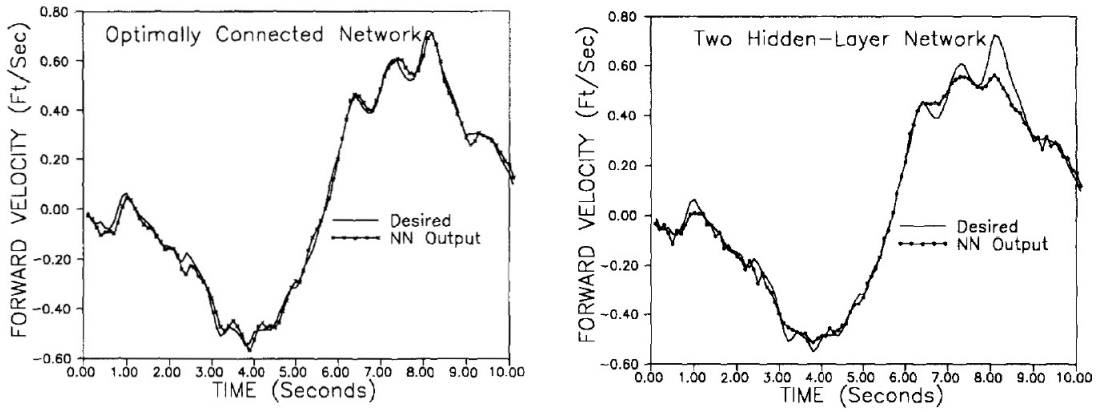


Figure 22: Forward velocity modeling comparison for the UH-1 helicopter model identification problem [49]

CHAPTER III

RESEARCH QUESTIONS AND HYPOTHESES

In the previous chapter, the brief observation in the current practice of the ANN surrogate modeling in the field of aerodynamic analysis and the inconvenient problems in the selection of the network architecture have been described. Then, through the observation on the leverage of the network architecture onto the modeling performance, the proper selection of the network architecture has been turned out as one of the effective means to improve training efficiency and accuracy. Also currently existing methods and ongoing efforts on the principled determination of network size and connection structure have been reviewed. Each method has its own strength and weakness in the context of the applicability to the surrogate modeling tasks of practical scale. In this chapter, the overarching question of how to design a proper neural network for the given modeling task is refined research questions.

3.1 How to Explore Topological Space?

Before pursuing the optimization of the neural network in terms of connection structure as well as connection weights, an arising question is about the basic mechanism of searching the topological space. Letting alone the topological optimization, there exist many possible choices in the searching mechanism for the weight optimization. For example, heuristic stochastic search such as the genetic algorithm can also be applied to weight optimization while the mainstream tools for this purpose are gradient-based optimization techniques. In many research papers [93], the attempts to exploit global optimization techniques for the weight optimization have been made and compared in their modeling performance and numerical efficiency to the local optimization methods. In other words, whether gradient-based or non-gradient-based search will be

used in the weight optimization is one valid question.

But, in cases for the topological search, this choice for the basic mechanism for search is quite limited within the fundamentally discrete *operations.* Majority of the evolutionary approach adopting TWEANN strategy directly encodes the network connectivity into an independent setting in the binary chromosome strings; 0 for absence of connection and 1 for connection establishment, which are permanently fixed during the learning process. Network pruning methods also operate discretely; eliminating or keeping connections depending on their saliency's or sensitivity's threshold. In the Cascade Correlation and the Projection Pursuit learning networks, this discreteness exists in the neuron level by adding a single neuron with the multiple connections as a unit of topological search.

This discreteness in the search mechanism might have originated from the nature of the connectivity between the computational units, i.e., whether connect or not any particular pair of neurons. *But the discrete nature of the connectivity does not necessarily constrain the search mechanism to be a discrete operation only.* One good example is the previously reviewed KrishnaKumar's approach, which successfully infuses the connectivity into the continuous functional form with the continuous connectivity parameters enabling gradient-based search algorithm to work efficiently. Hence, the more fundamental reason for the general scarceness of the *continuous* or gradient-based topological search might have originated from the implicitly underlying assumption on the role of the network learning and training, i.e., the learning practice for the given connection structure while, theoretically, a network can learn any information such as the connection structure from the external stimuli represented by the training samples.

3.1.1 Network Sizing Schemes

The constructive methods adjust network topology by adding connections or neurons after each *unit* learning process has been completed. And the newly added network components usually require the independent learning process such as the correlation maximization in the Cascade Correlation and the *back-fitting* process in the Projection Pursuit method. In this way, the overall learning procedure consists of multiple *unit* phases. In this manner, these methods adjust the network size by adding computational unit one by one rather than adjust network structure in the connection level. Therefore, they serve as a means to size the predefined, particular types of the networks rather than as topological search methods although the network grows to its proper size and, hence, it learns the valuable topological information. The basic learning mechanism in these methods is strictly devoted to the weight optimization for the given network connection structure which, in this case, grows in its size as the training proceeds.

3.1.2 Meta-Search for Topology

The applications of evolutionary algorithms or simulated annealing for the topological search of the ANN work based on the strict separation of weight optimization and topological search. For example, the simultaneousness in the optimization of both the connection topology and the weights in the TWEANN methods are usually true only in the macroscopic point of view. In general, the TWEANN methods evolve the population by genetically reproducing the connection structure and the initial weights for that structure. During the learning process, each and every individual network in the population of each generation is trained for the optimal weights. Hence, evolution and learning are realized in the different levels. Genetic operations determining network structure and following weight optimization for that structure is also a faithful instance within the conventional learning practice of the weight optimization for the

given network connection structure.

3.1.3 Weight-Connectivity Dichotomy

The majority of the current topological search methods are based on the *weight-connectivity dichotomy*, and this dichotomy is related to the conventional learning paradigm, which focuses on weight optimization while letting other network characteristics such as connection structure to be determined by judicious selection through rather unprincipled ways. As Happel pointed out [35], the lack of the basic mechanism to learn the connection structure is also the reason why the *unstructured* networks such as the MLP are the usual choices for the ANN applications.

But the weight-connectivity dichotomy is not a mathematically inevitable constraint. There might be several ways to execute continuous topological search for the connectivity as well as the weights towards reduced training error. The closest example in this direction found so far was, again, the KarishnaKumar's method [49]. The expected outcome of the continuous topological search mechanism or the connectivity adjusting learning scheme is the seamless integration of topological search and weight optimization by which the network training efficiency can be significantly enhanced. For example, if the learning process could optimize the network connectivity as well as the weight parameters, the need for the higher-level optimization loop such as evolutionary algorithm or simulated annealing can be avoided or the higher-level optimization can operate with significantly reduced dimensionality. Also, many architectural constraints imposed in the constructive methods can be relieved by letting training process itself determine its connection structure.

Interestingly, in the biological realm, neuroscientists are unifying the theories for synaptic weight changes and connectivity adjustments, which have long been regarded as separated phenomena. And the growing number of evidences strongly support that the activity-dependent mechanism adjusts the network's connection structure as well

as the strength of the synaptic connections.

3.1.4 Biological Observation

At present times and in general, the biological neural network is regarded as the resultant outcome of both the evolution and the lifetime experience, i.e., genetic information inherited from the parents largely determine the network structure and the instructions for the further development and the lifetime learning fine tunes the detailed structure and the individual synaptic connections. This overall mechanism is loosely implemented in the evolutionary approach for the ANN optimization; genetic operators adjust the network structure and its initial weights and the individual network learns for the optimal weights. But, recently, the role of the activity-dependent mechanisms on the formation of the connection structure is being revealed as more active than the conventional understanding in the field of neuroscience.

There has long been a debate polarized by the mutually exclusive two theories explaining the development of biological neural networks; Sperry's *chemoaffinity* hypothesis versus Hebb's rules of correlation-based synaptic change [16]. Sperry's *chemoaffinity* hypothesis stated that the specificity of mapping pre-synaptic and post-synaptic partners is determined by molecular cues favoring the genetically predetermined destiny for the connection structure while the Hebb's rules allow more degree of freedom for the network connectivity emphasizing the role of the lifetime learning. One crude interpretation is that, for these two mechanisms to coexist, connectivity has to be determined by the *nature* and synaptic strength has to be adjusted by the *nurture*, which seems to perfectly fit the conventional learning paradigm of the ANN.

But recent findings in neuroscience strongly support that the connectivity itself is significantly changed by the lifetime learning. Scelfo and Buffelli described the *plasticity* in the biological neural network as follows [42]; *"Experience throughout lifetime sculpt the synaptic connections in the nervous system. These modifications are*

particularly pronounced during the developmental period called the critical period that leads to a highly refined degree of neuronal connections, which characterize the mature stage. However, also in the adult nervous system, synaptic strength and connections maintain a certain degree of plasticity in order to ensure experience-dependent adaptation of the nervous system to environmental stimuli. The word plasticity has been applied to a wide variety of nervous system changes. It is used for changes in synaptic strength and in synaptic connectivity." Here, the most relevant match of the terms *synaptic strength* and *synaptic connectivity* in the ANN context are *connection weight* and *connectivity* in the network structure.

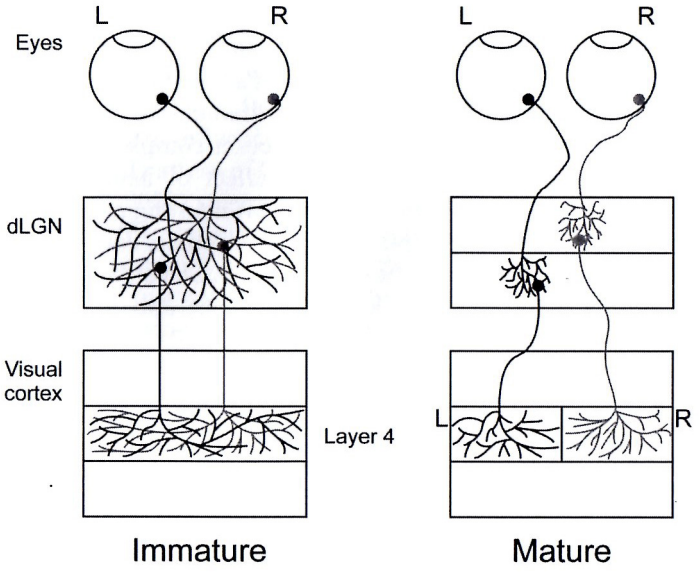


Figure 23: Axon pruning in the mammalian visual system [42]

Figure 23 compares the immature and the mature neuronal connections from the retina to the visual cortex, which shows the remarkable level of adjustment from virtually unstructured network towards the fine tuned structure.

Figure 24 summarizes an experiment result on the barn owls who are able to catch prey in the dark, solely based on auditory cues, after experience-dependent alignment of visual and auditory maps in the Central Nervous System (CNS) and learned motor output. The maps of auditory (marked A in the figure) and visual (marked V in the

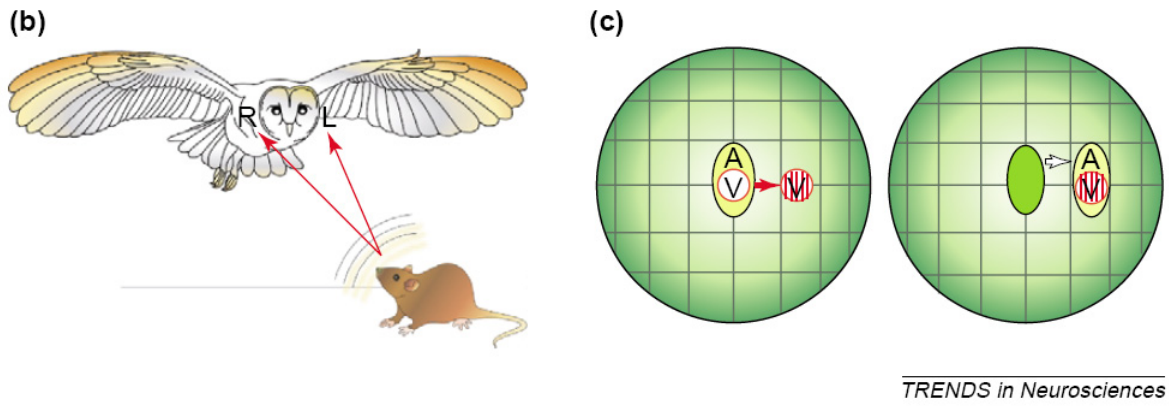


Figure 24: Re-alignment of barn owl's auditory maps (A) in the central nervous system following the change in the visual maps (V) [16]

figure) space are normally superimposed (left figure) but when juvenile owls are fitted with prisms to shift the visual world by about 20^o, the auditory map also shifts to re-align with the visual map (right figure). This realignment of the auditory map is a clear example of the activity-dependent rearrangement of synaptic connections [16]. In this case, the level of adaptation of the network connection variation is learning rather than evolution.

Although the entire business in the *artificial* neural network has been inspired by its biological counterpart, there is no necessity or relevance to be constrained or guided by the biological findings. For example, even the very basic concept of the supervised learning is hard to be comfortably fit in the frame of biological learning, i.e., there are no *teacher cells* explicitly providing the error signal propagated back, especially, following the exactly same path by which the forward signals propagated [37, 15]. Moreover, contrast to the allowance for the *positive* and *negative* weight values commonly used in the ANN, the biological neurons are known to be destined to either one type of *excitatory* or *inhibitory* neurons, i.e., majority of biological neurons does not emit mixture of positive and negative signals in case of crude translations from excitatory and inhibitory signals [17]. Recent findings in the experimental neuroscience show the potentially important role of the electromagnetic modulation in

neural communication beyond the electrochemical path [2]. This means that there might be another fundamental route in the basic information flows in the network other than one strictly depending on the physical connections between nodes. Therefore, at least for the time being, the biological plausibility in the artificial neural network is very hard to be claimed even in the macroscopic point of view. Bishop described once like this [9]; *"... many of which have been the subject of exaggerated claims regarding their biological plausibility. From the perspective of practical applications of pattern recognition, however, biological realism would impose entirely unnecessary constraints."* The superficial observation described in this section serves, at best, just *inspirational purpose only.*

3.1.5 Connectivity Adjusting Learning Scheme

For optimization of the weight parameters, there exists wide spectrum of working methodologies such as more conventional gradient-based optimization techniques and genetic algorithm. Virtually every general optimization techniques have been tried, refined, and applied to the problem of obtaining significantly better weight set than the random initial weights. But, on the other hand, for optimization of the network topology or the connection structure, only handful unique techniques have been devised and those are, in general, formulated outside the conventional optimization schemes resulting in the additional meta-search loop for topology or the stereotyped topological adjustment rather than exploring diverse connection patterns. However, the biological neural network changes, seemingly seamlessly, its connection structure as well as synaptic strength responding to the external environment although its initial layout has been imprinted in the organism's genetic codes.

Therefore, it is one reasonable way to investigate, first, the lowest-level learning mechanism for connectivity as well as weights of the network before attempting a contrivance for another sophisticated method under the limitation of the conventional

learning paradigm.

Not only the connectivity adjusting learning is natural for the simultaneous search for both the weights and the connectivity but also this approach has great potential to enhance training efficiency by creating sparsely-connected multi-layer structure without the need for the higher-level, meta-search for topology and the unnecessary or ambiguous constraints on the network architecture. In the conventional MLP networks, the way to implement additional layer of computational units requires significant amount of additional computational time and memory due to the *unstructuredness* intrinsic to the *fully-connected* regularity.

The concept of *connectivity adjusting learning* permits wide interpretation including any training method in which not only weight parameters but also connectivity themselves experience adjustment during the learning phase of the network such as in the evolutionary approach, the constructive and pruning methods. The more specific objective in the current thesis is to develop an integrated learning mechanism for connectivity as well as weights satisfying following distinctions from the more general category of this class of learning methods;

- It adjusts both the connectivity and weights within a single training campaign instead of two dissimilar optimization loops consisting of a topological optimization loop in the higher level and a weight optimization constrained by the given topological structure of the network.

- It does not require a series of re-training around the events of changes in the connectivity such as addition or elimination of network connection and/or computational node.

- It has to be suitable for learning procedure for the general multi-layer network architecture guaranteeing training efficiency comparable to the contemporary

training algorithm for the supervised learning tasks such as the Levenberg-Marquardt algorithm.

3.2 How are Weights and Connectivity Linked Together?

To develop a new learning mechanism adjusting both the connectivity and the weights, the basic understanding is required for the relations between these two dissimilar properties; the weight which is continuously varying and the connectivity which is strictly discrete in its nature.

3.2.1 Connectivity At Zero-Value Weight

Between two particular computational units or neurons, if there is a link or connection, the connectivity is established and the weight value can be assigned as an amplification factor of that particular connection. The direction of the information influx in this connection is one way from the lower node to the higher node assuming the notional feed-forward network architecture while the propagation of error flows in the reverse direction. Hence, there are two ways to make a particular connection inactive; one way is to simply *turn off* the connectivity and the other is to make the weight *zero*. The sensitivity-based network pruning methods employ the former mechanism for the post-training networks and the penalty function-based network pruning methods aim the natural occurrence of the latter during the training process. But these two states of current inactivity are not exactly identical in the effect on the further training process, i.e., if the connectivity for a particular connection is false then, the weight has to be interpreted as zero in the training process but not every zero-weight corresponds to false connectivity. For example, some of the weights have possibility to be zero at any point during the training process but, as long as the connectivity is flagged as true, these zero weights have chance to converge, later, to another finite values other than zero.

This fact gives a new interpretation for the absence of connections, i.e., the status

of no connectivity can also be viewed as just zero-value weight with true connectivity. As in the case of connection pruning, this new interpretation can have different effect on the further training results.

Therefore, there exists the interchangeability in status quo between the connectivity and the weight value for a single connection when the connection has a zero-weight value but the effect of their exchanges will cause networks to learn differently. By revisiting the BP algorithm, it will be shown that the effect of this exchange on the network training can be calculated exactly and efficiently.

3.2.2 Revisiting Back Propagation Algorithm

At any arbitrary network status during the training using the BP algorithm, each and every computational unit or neuron updates two properties, at each training epoch, reflecting external stimuli.

- A forward propagation of input data passed through all the previously located and connected neurons;

$$z_i = g_i(\sum_h (w_{ih} z_h) + b_i) \tag{29}$$

  where z is the activated value of each neuron, g is the activation function, w_{ji} is the weight connecting arriving to neuron j from neuron i, and b is the bias of each neuron.

- A backward propagation of output error passed through all the rear located and connected neurons;

$$\delta_j = g'_j(\sum_i (w_{ji} z_i) + b_j) \sum_k \delta_k w_{kj} \tag{30}$$

Here, the first property has been expressed for neuron i and the second property for neuron j. Simply multiplying these two terms results in the derivative of error

with respect to the weight, $\frac{\partial E^n}{\partial w_{ji}}$, connecting from neuron i to neuron j as derived in the previous chapter. Obviously, this weight gradient exists for each and every pair of neurons in the network regardless of whether there exists an established connection between them or not. And, if the multiplication is executed, the calculated weight gradient for the absence of the connection corresponds to the derivative of error with respect to the zero-weight value. In other words, the standard BP algorithm provides enough information to calculate weight gradients for all possible feed-forward connections between the given numbers of the neurons even though the network is only sparsely connected. This means that instant calculation of error gradients of currently absent connections can be executed to measure the effect of possible conversion from false connectivity to true connectivity with zero-value weight during the error minimization process. Obviously, this aspect has not seriously been exploited in the conventional ANN implementations.

3.2.3 Invisible Connections

The beauty of the BP algorithm is in its generality in the sense that it doesn't assume any constraint on the network's detailed structure except the feed-forward nature of the connections'. Single sweep of forward- and back-propagation results in calculated z and δ terms for each and every neuron in the network providing enough information to calculate the error gradient for any feed-forward connection weight regardless of whether it is present or absent in the current network. Here, an *absent* connection can be interpreted as a *present* connection when we include zero-weight connections in a set of valid connections. These zero-weight connections are designated as *latent connections*. The gradient information of the latent connections can be used to establish new connections in the same way as the weights are adjusted depending on their gradient values to reduce training error. For this, one algorithmic element additional to the standard BP algorithm is to calculate the weight gradient

of each and every feed-forward connection even if it is not currently valid connection. Assuming a *batch* training process in which entire N-training samples are used to determine the adjustment in the network such as used in the LM algorithm, the sum of individual sample's error gradients, *total* error gradient,

$$\frac{\partial E}{\partial w_{ji}} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial w_{ji}} = \sum_{n=1}^{N} z_i \delta_j \tag{31}$$

can be evaluated for any feed-forward connection. In OBG, the latent connections whose total error gradients are greater (in absolute magnitude) than those values of existing connections are promoted to be established as new valid connections. A new connection has initial weight value of zero and, then, the weight optimizer optimizes its value as training proceeds. Because of initial zero-weight for a new connection, inserting a new connection can be processed *seamlessly*, i.e., without any discontinuous change in resultant training error at the moment of making a new connection (Figure 25). Of course, without a proper regulatory mechanism, the network which is allowed to grow its connections in this manner will boundlessly grow towards the maximal connectivity, i.e., Fully-Connected Cascade (FCC). Therefore, an efficient exploitation of latent connections requires a balancing mechanism involving network pruning.

3.2.4 Error Gradient of Invisible Connections

As shown in the previous section, BP algorithm already provides the weight-error gradients not only for the present but also for the absent connections between each and every pair of computational units although the latter information is usually discarded in the conventional learning practice, which is hypothesized to be crucial to design a variable connectivity learning scheme.

By interpreting the instance of the absence of connection as a physical connection with zero-valued weight whose error gradient is readily available by error back propagation algorithm, the partial information on the geometry of the weight-error

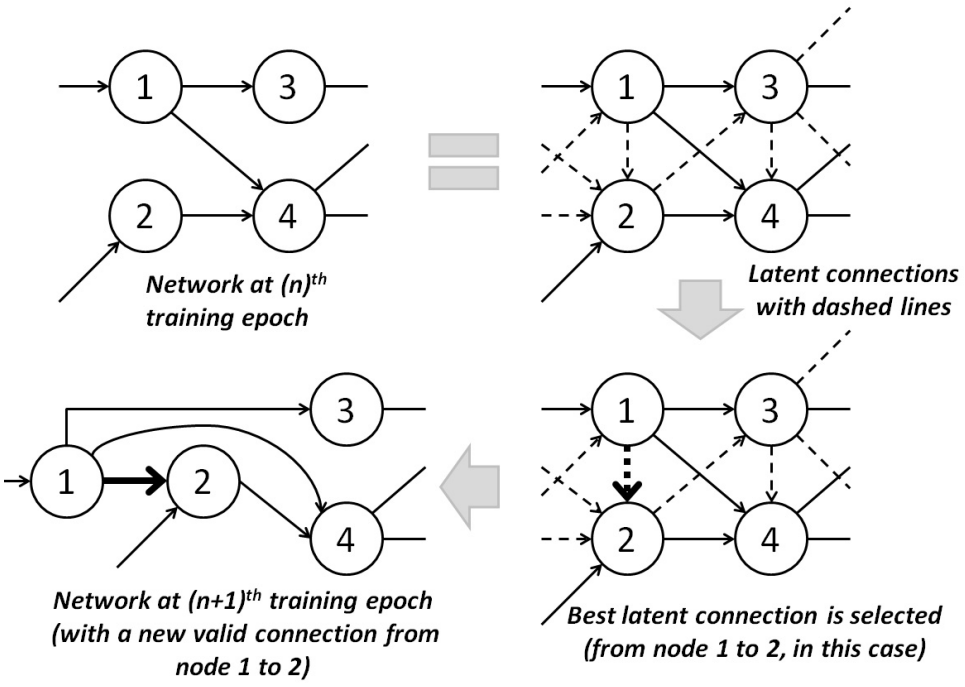


Figure 25: An example of the connection growth based on the concept of the latent connections

space in conjunction with the new connectivity is obtainable. This instant snapshot might be a critical piece of information to facilitate the variable connectivity learning scheme because it enables quantitative estimation of the reduction in training error corresponding to an introduction of the new connectivity between particular pair of neurons like we estimate the error reduction corresponding to the particular weight adjustment.

Using the weight-error gradient for a new connectivity at its zero-value weight, the proper direction of weight adjustment to reduce error can be determined and, hence, any additional connectivity will reduce training error while the magnitude of reduction in total error (for all output nodes and for all training examples) is not guaranteed to be larger than the case without introduction of the new connectivity. In general, if the network is not excessively large, the more connections usually result in the more accurate training result by increasing network's memory capacity. Of

course, this is not always an advantageous phenomenon because increment of connections requires more computational time and memory for their training. Therefore, to prevent inefficient growth in network connectivity, it is necessary to devise an appropriate mechanism, which determines whether any particular new connectivity has to be established or not.

This hypothetical strategy exploiting the *hidden* gradient information for currently absent connections has potential to be augmented to an integrated learning mechanism adjusting connectivity as well as weights in conjunction with additional logic for the choice for the particular set of zero-weight connections to grow.

One peculiar aspect of the *hidden* gradient information is that the gradient for a particular connection which has no connectivity has non-zero value only when connecting two nodes are effectively working nodes in the network, i.e., when the lower index node is connected to at least one of the input nodes and the higher index node is connected to at least one of the output nodes. Therefore, the existence (corresponding to non-zero value) and the magnitude of the *invisible* connections' weight-error gradients depend on the current network configuration and this dependency is traced back to the initial network status.

In this way, the pursued connectivity adjusting learning scheme's efficacy will be bounded as a local learning method whose searching ability has to be compared with other conventional local learning methods such as the BP-LM method.

3.2.5 Growing and Pruning

Unfortunately, the hypothetical strategy is appropriate only for growth of connections toward bigger network not providing any clue about how to prune less necessary connections.

Inclusion of pruning mechanism into a notional connectivity adjusting learning scheme is harder to achieve than the growing approach. We have seen the possibility

of *seamless growth*, i.e., always in error-reducing direction, by adding a new connection in this section but the only possible way of eliminating connection *seamlessly*, i.e., without deteriorating training performance, is to naturally drive weight to be zero adopting penalty term-based pruning methods. But penalty term-based pruning mechanism has a problem to be combined with the most efficient optimization scheme, the LM algorithm because the LM algorithm specifically working on minimization of the squared-error type function which needs significant variation in its functional form with added penalty terms. More fundamentally, as pointed out by Reed [70], penalty term tends to favor weight set with many small weights over a single large-valued weight, even when this is more effective choice. This behavior will compensate with the decreased number of connections.

Sensitivity-based pruning approach is also not suitable for the current objective to design a new learning method because it operates on the post-training networks. The attempted new learning method is expected to learn the better connectivity as well as the better weights within a single training campaign without the need for the re-training process.

If the network is not big enough to cause a significant over-fitting problem, pruning methods are for minimizing impact on the present net status in case of reducing the number of connections and, hence, not affecting network's generalization performance positively. In this sense, the growing strategy is more appropriate in the attempt of enhancing network's generalization performance as learning proceeds, especially, when the learning starts from a small-size network.

Therefore, the disadvantage of being one-directional growth has to be carefully overcome by starting learning from the sparsely or minimally connected network and avoiding over-fitting problem as the network grows.

3.3 Which Connections to Grow?

The effect of conversion from the absence of connection to the zero-weight connectivity can be quantitatively calculated by the BP algorithm in the form of the derivative of error with respect to the weight variation. As soon as this conversion occurs, the corresponding weight will adjust its value to non-zero value as the learning proceeds. Assuming starting from a sparser network than the fully-connected cascade, a particular set of connections has to be chosen to grow towards the optimal network for the given training data.

3.3.1 Random Selection

Based on the fact that any additional connection will contribute to reduce training error by initially having zero-value weight, one possible strategy is to randomly choose some of absent connections to grow from a zero-weight value at each training epoch. In this case, the number of new connections at each growth event and the rate of this event occurring (i.e., in terms of per training epochs) have to be determined. This strategy does not need to calculate the gradient information for those *hidden* connections. Besides the ambiguity in determining the appropriate *growth rate*, this strategy is susceptible to unwieldy growth towards fully-connected cascade.

3.3.2 Gradient-Based Selection

If we exploit the *hidden* gradient information to determination of the next new connection on a particular network status in learning, more reasonable outcome than random selection is expected. For example, instead of the random selection, we can choose the *hidden* connection whose error gradient is the largest in its magnitude among all existing *hidden* connections. Of course, the *growth rate* for the selection has to be determined in this approach, too. Before dealing with the generalization performance, a network training process is, basically, error minimization process. The addition of new connection(s) whose error-reducing effect is maximum among

all currently absent connections at the current network status and searching the best possible weight set using an optimizer is one step closer mechanism to the current aim of developing basic mechanism adjusting connectivity as well as weights. But this approach also does not guarantee to enhance the numerical efficiency of training because it is based on simple increment in weight parameters. The reason why the conventional MLP networks are preferred than the fully-connected cascade (FCC) architecture which has the maximal connectivity for the given number of neurons in majority of the supervised learning tasks is that in case of the satisfactory training performance, the training efficiency of the MLP networks in terms of time and memory usage is far better. Therefore, the training efficiency is an important criterion for devising a new learning scheme.

3.3.3 Consideration for Training Efficiency

In the conventional learning paradigm in which the network learns the error-minimizing weight set for the given connection structure, there is no degree of freedom to directly control the computational resources, i.e., the number of optimization parameters or the number of the connections and biases are fixed and the course of the weight adjustment is strictly dictated by the operation of a numerical optimizer. But extending the scope of learning to the connectivity in the network makes the computational time and memory required for each epoch of the training depend on the learning trajectory affected by adjustment of the number of the connections and by the rate of growth. The previously described selection strategies for new connections correspond to allocating monotonically increasing computational resources by keeping adding new connections at a given growth rate. Contrast to the previous strategies, maintaining the initially allocated level of computational resources throughout the entire training process results in another selection criterion to determine which *hidden* connection to grow;

- To make a new connection whose weight-error gradient is the largest in its magnitude among all possible pairs of unconnected neurons *only when* that magnitude is greater than the smallest magnitude of the weight-error gradients of the present connections

- Then, to keep the weight of the connection whose weight-error gradient is the smallest in its magnitude *frozen* as a fixed constant for the further training epochs

In this strategy, the number of weight optimization parameters is always kept as constant and, hence, the time and memory required for weight optimization at each training epoch is maintained as approximately the same as the initially allocated ones. Therefore, the problem of the appropriate network size is partially converted to the problem of how to determine an appropriate initial allocation of optimization resources, i.e., the number of adjustable parameters, which will be fixed throughout whole training process in case of no additional computational units. Again, the dependency of the network performance on this property might be inevitable due to the locality of the proposed scheme but this does not severely constrain the size of the network by allowing arbitrary growth in the connectivity because although the number of optimization parameter has been fixed as the given initial network structure, the effective number of model parameter grows depending on the gradient information for whole connectivity.

This strategy of partial optimization is a big departure from the conventional weight optimization practice in which all the weight and bias parameters are usually optimized at each and every training epoch. The only witnessed case adopting partial optimization scheme among all possible free weights and biases is in the Cascade Correlation method which *freezes* all the previously trained weights at the event of addition of a new hidden unit and optimizes newly added connection weights only

until another hidden unit is added to the network.

Another unconventional notion comparable to the current approach of partial optimization is found in the Le Cun et al.'s work on the Optimal Brain Damage [52]. In their paper, following description is given for their hand-written zip-code recognition case which has been reviewed in the previous chapter [52]; *"The simulation results given in this section were obtained using back-propagation applied to handwritten digit recognition. The initial network was highly constrained and sparsely connected,* ***having*** 10^5 ***connections controlled by*** 2578 ***free parameters.*** *It was trained on a database of segmented handwritten zip-code digits and printed digits containing approximately* 9300 *training examples and* 3350 *test examples."* This means that the actual connections existing in the networks are *controlled* by far fewer parameters.

3.3.4 Optimal Brain Growth

The hypothesized growth rule builds a new connection only when that new connection has the steeper weight-error gradient than, at least, one of the existing connections'. This results in dynamically maintaining a pool of relatively higher weight-error gradient connections as optimization parameters and this relative measure is especially meaningful when we consider that the allowable computational cost (including time and memory consumptions) is finite and has to be minimized eventually. The advantage of this rule is that it manipulates connectivity towards accelerated training process in computationally efficient manner because it directly exploits a measure of how steep the weight-error gradient of a potential new connectivity is and does this while keeping the same number of optimization parameters for the weight optimization.

This particular learning strategy built upon the hypotheses presented so far has been named as *Optimal Brain Growth* in the similar level of metaphorical abstraction with *Optimal Brain Damage* or *Optimal Brain Surgeon* in the realm of strictly

artificial neural networks. At this point, it is a purely hypothetical concept rather than a thoroughly formulated algorithm, and the mathematical formulation and computational implementation will be completed in the next chapter after conceptual augmentation to have a form as a definitive learning method.

In the conventional learning paradigm, the weights are adjusted strictly in the direction of reducing training error for the permanently fixed network connection structure. In the notional Optimal Brain Growth learning, the weights are also adjusted to reduce error exploiting their gradient to error and the new connections are grown out of having zero-value weight in place of slowly changing, hence, more stabilized connections in their weight variations making them excluded from the next optimization process. Of course, these excluded connection weights from the current optimization process will be re-included depending on the distribution of the weight-error gradients in the later training epochs. In this way, the number of total connections in the network might increase or remain the same in case of re-selecting the previously excluded connection weight instead of growing a new connection depending on the distribution of *hidden* or *present* weight-error gradients at each training epoch. This basic mechanism has great potential to facilitate a seamlessly integrated learning for connectivity and weights without demanding more computational time and memory at each training epoch.

3.3.5 Growth Control

In general, the best training error which is obtainable through the weight optimization process monotonically improves as the size of the network increases by extending its *memory* capacity. Therefore, the optimal size of network is only defined when the other criteria than the obtainable minimum training error is introduced. One common choice for this criterion is the minimum validation error for which a separated set of validation samples are required [70]. Depending on the nature of the given problem,

there also exist other critical constraints such as the available maximum training cost or the allowable maximum network size additional to the satisfactory training and generalization performances. In any cases, it is important to suppress inefficient growth restricting a network to grow only when it is *necessary*. In OBG, this *necessity* is defined by the following growth control rules.

3.3.5.1 Magnitude-Based Hard Pruning

It is possible to maintain the number of connections unchanged around connectivity adjustments by establishing a new connection only when an old connection is pruned. During the weight optimization process, a connection weight may reach close to zero value making that connection's contribution to the network output negligible, at least temporarily, at that moment. Pruning this near-zero-weight connection and making the best latent connection (whose total gradient value is the maximum among all the latent connections in its magnitude) as a new valid connection is one rational set of connectivity adjustments which guarantees the seamless variation in the training error. The near-zero-weight connection has to be included in the set of latent connections immediately after it has been pruned, before the selection for the best latent connection. In this way, the usual negative-to-positive or positive-to-negative weight changes are allowed in the form of pruning and creating the same connection. Otherwise, the connection structure of a network is modified as the result of pruning of an old connection and creation of a new connection. All these structural modifications occur faithfully guided by the gradient-based weight optimization process. This growth control method is particularly useful to assess the benefit of connectivity adjusting learning in comparison with the conventional, weight-only learning with a fixed network structure because it eliminates the advantage of the larger networks in the aspect of the best obtainable training error by maintaining the same number of connections in the two learning cases sharing the identical initial network.

3.3.5.2 Freezing Stabilized Connections or Soft Pruning

In the higher-than-first-order optimization such as the LM algorithm, the most resource-consuming step is the inversion operation for the Jacobian matrix and the computational cost increases as the size of the matrix grows. Considering that the size of the Jacobian matrix is solely determined by the number of optimization parameters, the additional training cost incurred from network growth can be significantly reduced by maintaining the number of *optimizable* connections unchanged. This can be achieved by *freezing* an old connection weight and relieving it from the valid set of optimization parameters whenever a new zero-weight is introduced as a new optimization parameter. The *partial optimization* strategy in the neural network training was introduced earlier such as in Fahlman and Lebiere's work on the Cascade-Correlation as an attempt to overcome the inefficiency in training layered networks at that time [52]. To facilitate this type of operations, each and every connection weight has to be equipped with an *optimization flag* which indicates whether the corresponding weight value is variable or fixed one at a certain point in the training process. The straightforward procedure adopting this strategy is

1. Detect one existing connection whose weight has been *stabilized*, i.e., whose weight shows very small variation under the pre-defined numerical threshold consecutively during the pre-defined number of training epochs

2. Select one latent connection whose total gradient is the maximum in its magnitude

3. Set the stabilized connection as a latent connection which is not optimizable and make the selected latent connection as a new optimizable connection

The processed stabilized connection can be reselected as an optimizable connection applying the same selection criterion. In this way, a selective weight optimization

is executed only for the dynamically maintained pool of high-gradient connections. By searching the more effective weights in training error reduction and concentrating optimization resources on them, learning process is accelerated without significant increase in computational cost, resulting in a task-adapted connection structure.

3.3.6 On Over-Fitting Problem

At this point, the notional Optimal Brain Growth seems to learn connectivity and weights, which reduces training error further at any arbitrary network states by the mechanism exploiting *whole* gradient information including from *present* and *hidden* connections. And at the event of connection growth the number of adjustable weight and bias parameters is maintained as the same, which is the basic mechanism to maintain training efficiency. The synergy of these two mechanisms is expected to accelerate the training process, but the evident increment of the number of total connections has possibility to deteriorate, by adding the degrees of freedom to the model, the generalization performance, which is the ultimate goal of every ANN training. Because the training process is explicitly governed by the training error only, it is difficult to add any explicit mechanism aiming the direct enhancement of the generalization performance. The most relevant symptom of the deterioration of the generalization is the occurrence of the over-fitting problem.

Figure 26 is a notional schematic diagram for the variation of notional training error and validation error. In the early stages of training, both the training and validation error tend to decrease as learning proceeds, but at some point, the validation error representing the generalization performance reaches a minimum and begins to increase. This is because the usual training data is incomplete and it contains spurious and misleading regularities depending on the sampling [70]. Hence, one common approach to avoid *over-training* or over-fitting is to estimate generalization capability by employing separate set of validation data and stop the training process earlier

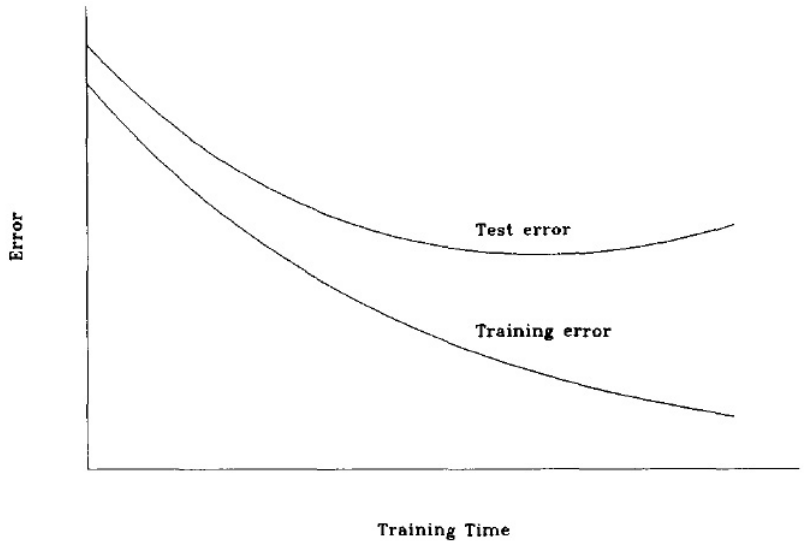


Figure 26: The notional schematic diagram of training and test error [70]

than the minimum training error is reached, at the minimum validation error point. Another heuristic way of avoiding over-fitting is to use less flexible network model by constraining its degrees of freedom (the number of connection weights and biases, roughly) because over-fitting is thought to happen when the network has more degrees of freedom than the number of the training samples [70], which is also the basic consensus behind network pruning methods.

Because the proposed growth mechanism might deteriorate the generalization performance more severely than the case of non-growing learning mechanisms by monotonically increasing the number of model parameters, the learning has to begin with a minimal, sparsely connected network which has relatively small degrees of freedom and has to employ the early stopping strategy using the separated validation set.

3.4 How to Add a New Neuron?

As reviewed in the previous chapter, for the given training data samples, the network with too small number of neurons is unable to learn the underlying nonlinearity among the data, and the network with too large number of neurons not only causes

computational inefficiency but also has inferior generalization capability. So far, the Optimal Brain Growth has been conceptualized for the fixed number of computational units allowing the growth in connectivity only. But the number of possible connections in the network is always governed by the number of the computational units.

Therefore, the determination of the appropriate number of neurons and the growth rule to appropriately add a new neuron to the network is an important issue. But the similar difficulty to detect the need for an additional computational unit exists as the case for an additional connectivity, i.e., an additional computational unit always results in the reduction of training error even though it corresponds to a net loss in the generalization performance in the final training result. Moreover, contrast to the efficacy of a new connectivity which can be estimated by the gradient information, the efficacy of a new neuron is basically unable to be estimated using the BP algorithm except by trial-and-error approach such as in the dynamic node creation repeating whole training procedure when a new component added to a network.

Excluding time consuming re-training approach, one possible strategy is to train network fully and, in case of unsatisfactory prediction performance, add a neuron and train the network further. This approach has some similarity with the way new hidden nodes are added into the network in the constructive methods such as the Cascade Correlation and the Projection Pursuit learning networks. But in these approaches, both the CC and the PP add a *fully-connected* neuron whose connectivity has been strictly pre-determined by their network structure, i.e., the fully-connected cascade (FCC) and the single-hidden layer network instead of searching for the optimal connectivity. *Further training* approach without re-training needs a mechanism to assign proper *initial* weights for the newly added computational units and the maximization of correlation of the input weights for the newly added neuron in the CC and the isolated optimization of the connections and activation function in the PP serve to that purpose.

In the Optimal Brain Growth, there is no need for adding a *fully connected* computational unit because the appropriate connections can be grown even though there is no connections initially as long as the newly added node has the minimal connection to the network. This characteristic provides a natural way to extend the size of the network by introducing a new computational unit having only sparse connections. In the OBG, either of the following two mechanisms can be used.

3.4.1 Adding a New Neuron

- When there is a need for adding a computational unit such as stagnated training performance, etc., add a single neuron which has the minimal connectivity, i.e., one input connection to the newly added neuron and one output connection from that neuron.

- A new output connection is defined as a connection from the newly added neuron to the output node which has the greatest training error among all output nodes.

- A new input connection is defined as a connection to the newly added neuron from the computational unit, which has the greatest correlation value between its node output and the maximum error output node as defined previously. The definition of *non-dimensionalized* correlation which is from the Cascade Correlation method [24] is, originally,

$$S = \sum_{o} | \sum_{p} (V_p - \bar{V})(E_{p,o} - \bar{E}_o)| \tag{32}$$

  where S is the sum over all output units o of the magnitude of the correlation between V, the candidate unit's value, and E_o, the residual error observed at unit o, which is the network output at which the error is measured and p is the training pattern. The quantities $\bar{V}$ and $\bar{E}_o$ are the values of V and E_o averaged

over all patterns.

The modified version for the maximum error node, E_{max}, is

$$S = |\sum_{p}(V_p - \bar{V})(E_{max} - \bar{E_{max}})| \tag{33}$$

where V is the activation value of the candidate input unit which will be connected to the newly added neuron with a single connection.

- In this formalism, as the result of an addition of a new neuron, the three optimization parameters are added to the network; the input weight value (connecting the maximum correlation neuron to the new neuron), the output weight value (connecting the new neuron to the maximum error output node), and the bias value of the new neuron. They are notionally assumed to have small, random values except the output weight value, which is assigned to be zero to enable seamless integration of the new computational components into the existing network structure.

- Under the feed-forward connectivity constraint, the index of the newly added neuron has to be greater than the maximum correlation node and be less than the output nodes. This adds additional randomness without explicitly determining the criterion for the relative position of the new computational unit.

In this way, when the network requires an additional computational unit, although the definitive criterion for this requirement is not provided yet, the network detects the most *troublesome* output node and the most relevant effector node for that output node, and connects them via a single randomly weighted connection, a new computational unit having a random bias, and a zero-weight output connection. As described above, this approach has been particularly chosen as the way of adding a minimal structure including a new computational unit, which has a maximal impact in the

training performance. After this event occurs, the Optimal Brain Growth learning is expected to advance optimizing weight parameters and building appropriate new connections around the newly added network components.

3.4.2 Decomposing an Existing Neuron

When the network converges to local optimum with unsatisfactory training performance, increasing the number of neurons can be considered as one way to enhance learning capability of the network. In the constructive methods, addition of each new computational unit is completed with total re-training such as in the Dynamic Node Creation [3] or uniquely defined sub-training processes such as the correlation maximization in the Cascade-Correlation [52] and the back-fitting procedure in the Projection-Pursuit [44] to facilitate functionality of the newly added unit and connections around it. In OBG, the intrinsic capability of connectivity adjustment significantly reduces the need for additional sub-training process and, instead of being re-trained from the scratch, the network which has been trained so far is *refined* by decomposing an existing neuron into two neurons for further training.

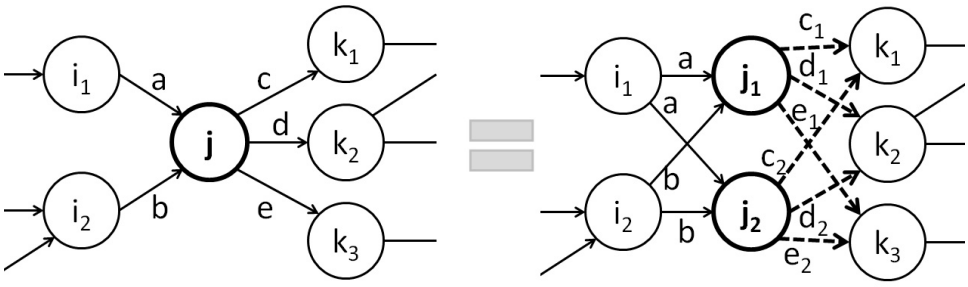


Figure 27: Two networks are identical in terms of the final network outputs in case of $c_1 + c_2 = c$, $d_1 + d_2 = d$, and $e_1 + e_2 = e$

The decomposition mechanism in OBG has been devised for two objectives. The first is the conservation of the learned knowledge on the problem up to the moment of decomposition and the second is the promotion for the escape from the local optimum. Observing that the convergence of network status to local minimum is a cumulative

phenomenon of converged output values of individual neurons, the *stabilized* neuron is searched at each and every training epoch and is decomposed into two-parallelly positioned neurons. The criterion for detecting stabilized neuron is as same as for the case of stabilized connection except that they are applied for the activated output values of the neurons. At each training epoch and for a particular training sample, the relative variation of each neuron's output value is measured and, for each neuron, all of them are averaged over the entire training samples. The selected hidden neuron is decomposed by the following rules (also shown in Figure 27);

- Two decomposed neurons have identical input connections to the original neuron's in their connectivity and weight values
- Two decomposed neurons have identical bias values to the original neuron's
- Two decomposed neurons have identical output connections to the original neuron's in their connectivity only and have randomized weight values whose sum for the same output neuron is identical to the original neuron's

Preserving the input connections and bias values during the decomposition event results in the identical outputs of the two decomposed neurons, which is also unchanged from the original neuron's. And, for the each destination neuron to which the original neuron connects, making the sum of output weights from the two decomposed neurons identical to the single weight value from the original neuron also contributes to the unchanged net effect of the decomposition process to the final network outputs. What has been changed is the landscape of error gradient field which has been altered by the newly added random weights in the decomposition process. Hence, this decomposition process results in *local reinitialization* strictly maintaining the final network output as before. Once decomposed, the new connections may grow or the stabilized connections may be pruned (in the hard or the soft manner) around the

newly added hidden units following the previously defined connectivity adjustment mechanisms.

3.5 How to Train a Growing Network Efficiently?

By casting this research question, the solution which is sought is limited in the context of how to minimize the increment of computational time and memory accompanied by the growth of networks rather than finding a fundamentally different way of numerical optimization. As long as residing in the gradient-based weight training scheme such as BP-LM method, the overall numerical efficiency of the network training is bounded by those algorithms' which is, in general, known as the best available among the variety of methods. One additional relevant factor for the training efficiency of the proposed Optimal Brain Growth is how to manage computational resources in the event of connection growth and node addition. In this section, the reason behind the partial optimization scheme and the minimal increment of connections in case of addition of the computational unit proposed in the previous section, is discussed in detail, especially, the training efficiency of the proposed growth algorithm is compared to that of the conventional MLP networks'

3.5.1 The Problem with MLP Learning

Hinton, one of the originators of the BP algorithm, explained the reason why many researchers immigrated from the MLP to the Support Vector Machine (SVM) in the pattern recognition and machine learning field in 1990s as the lack of scalability in the MLP network training [41]; *"The learning time does not scale well; It is very slow in networks with multiple hidden layers."*, despite of the superior learning ability of the MLP compared to the SVM's which he expressed as *a clever perceptron* in the sense that it has only limited adaptability to the training data. As clearly pointed out here, the *unstructuredness* of the conventional MLP networks which assumes regularly or fully connected hidden units constrains the enhanced training efficiency. In this

aspect, the proposed Optimal Brain Growth method is expected to possess enhanced training efficiency as the problem size grows because it builds the sparser, optimal connection structure for the given number of the computational units.

Fahlman and Lebiere also discussed in detail on the training inefficiency of the MLP networks and they *blamed* two major sources for that [24].

- *The step-size problem* occurs when the BP algorithm is used as a learning rule without being harnessed by more efficient optimization tool because, practically, only very small learning step guarantees for the weights to converge to local minimum. Introduction of momentum term and many algorithm exploiting dynamic learning step and, eventually, the more efficient second-order optimizers have been applied to cure this problem.

- *The moving target problem* occurs by the error signal which a particular hidden unit sees changing constantly. Units in the interior layers are given with the information as both the upstream and downstream units adjust their connection weights and biases, and this makes it difficult for such units to adjust decisively toward a good solution. In other words, instead of being stabilized quickly toward appropriate weight and bias values, each and every optimization parameter perform a *complex dance* taking a long time to settle down.

The fixation of the previously optimized parameters and, then, optimizing newly added components only in the Cascade Correlation method has been devised to prevent the moving target problem. And this strategy, in conjunction with the correlation maximization technique, results in the rapid convergence rate even though with the extremely, fully-connected FCC architecture. Therefore, the particular strategy *'to allow only a few of the weights or units in the network to change at one, holding the rest constant'* [24] which has also been adopted in the Optimal Brain Growth will contribute to acceleration of weight training procedure promoting stabilization

of optimization parameters.

3.5.2 Computational Bottleneck in Second-Order Optimization

Obviously, the growth of a network in the number of connections and computational units requires more computational time and memory for the propagation of input signal and the calculation of derivatives of error with respect to the weights and biases even though maintaining the same number of adjustable parameters. But this amount of increment in computational time and memory is negligible, at least in the problems of practical scale, compared to the most resource-intensive step in the second-order optimization, i.e., the matrix inversion operation of the Jacobian matrix. For example, as shown in the previous chapter, the Levenberg-Marquardt algorithm requires the right-hand side of the following equation has to be calculated to determine next weight set at every iteration,

$$\mathbf{W_{new}} = \mathbf{W_{old}} - (\mathbf{Z^T Z} + \lambda \mathbf{I})^{-1} \mathbf{Z^T} \epsilon(\mathbf{W_{old}}) \tag{34}$$

The Optimal Brain Growth maintains the dimension of the matrix to be inverted as constant value of (total number of adjustable weights and biases)^2 throughout the training process in case of not involving additional computational units. And in case of any additional neuron, there will be three more parameters (for an input connection weight, a bias for the new neuron, and an output connection weight) per each additional computational unit, which is a reasonable minimal amount compared to the fully connected weight set such as in the CC and the PP trainings. Therefore, by maintaining the same number of variable weight parameters for the given number of the computational units, a second-order optimizer such as LM algorithm can operate without significant increase in computational time and memory consumption in the proposed approach.

3.5.3 Robustness to Random Initial State

Another important issue relevant to overall training efficiency is the dependency of the final training results on the initially given network states. The appropriate initialization of the ANN is an important issue and there exist variety of numerical schemes to serve this purpose [40, 54, 60]. The Nguyen-Widrow method [60] provided the fundamental algorithm in this direction of approach. But, practically, the impact of the principled initialization scheme to the general training performance is depending on the problem at hand and even the majority of these approach, which fundamentally are responsive to the training data and the network architecture, has a great deal of randomness in the resulted set of initial weights and biases. Because the required number of training campaigns depends on the sensitivity of random initial values in the optimizable parameters, the robustness of any learning scheme in this aspect has to be investigated to estimate overall training efficiency.

3.6 Summary

In the practical surrogate modeling task using neural networks, the most common, *workhorse* network architecture-training algorithm pair is the Multilayer Perceptron (MLP) and the BP-LM algorithm. Despite of their wide applicability and satisfactory performance proven in the variety of fields, further enhancement through the principled approach towards the automatic network design still remains as an active research subject.

Based on the review on the way the contemporary methods attack this issue, the concept of *connectivity adjusting learning* has been hypothesized emphasizing following characteristics aiming further enhancement in overall training efficiency by concurrent learning both for connectivity and weights of the network;

1. Seamless integration of learning mechanism for connectivity as well as weights

where *'seamless'* means the realization of the monotonic reduction in the training error implying no need for the re-training process

2. Compatibility with the efficient optimization algorithm such as the Levenberg-Marquardt algorithm

The particular relations between the instance of absent connection and the physical connection of zero-weight value hinted the way of possible reciprocal re-interpretations and, in this context, the conventional BP algorithm has been proven to be able to provide derivative of error with respect to the weight of connection linking an arbitrary pair of computational units in the network regardless of the existence of established non-zero weight connection between them.

Exploiting this property in the conjunction with a particular, comparative criterion on the additional connectivity resulted in a notional connectivity adjusting learning scheme, the *Optimal Brain Growth.*

The particular selection criterion for the growth in the network connectivity is to always choose the higher-gradient-weight connection among the *hidden* connections keeping the relatively lower-gradient-weight connection among the currently existing connections frozen with the constant weight optimized so far and, hence, relieving it from the further weight optimization. In this way, the optimization process adjusts dynamically maintained pool of higher-gradient parameters only, maintaining overall training efficiency comparable to the non-growing network training even with the growing number of network connections.

The addition of new computational unit is also executed following the strategy to minimize the additional computational resources and to further exploit the appropriate growth in the network connectivity.

This extension from the conventional learning paradigm which focuses primarily on the weight optimization for the given network structure is expected to facilitate

principled ways of network construction without significant increase in the computational time and cost required in the training for the given data samples.

CHAPTER IV

OPTIMAL BRAIN GROWTH ALGORITHM

The Optimal Brain Growth is an algorithm encapsulating the hypotheses developed in the previous chapter. Basically, it is extension from the standard error back-propagation algorithm to exploit the derivatives of error with respect to the weight changes for *whole* weight set. Here, *whole* weight set corresponds to the set of the weights of all possible feed-forward connections including currently non-connecting *hidden* connections. In general, the weight values for *hidden* connections are zero but their gradients are not. In this chapter, the numerical algorithm for calculating the *hidden* gradients and, based on those values, the growth rule for a new connection is described in conjunction with the Levenberg-Marquardt optimization algorithm. Finally, the growth rule for a new computational unit is formulated. Combined algorithm as a whole constitutes a complete training method for a general feed-forward, Multi-Layer neural networks simultaneously learning optimal connectivity as well as weights.

4.1 Extension of Error Back-Propagation Algorithm

4.1.1 Definition of a Network

A network consists of finite number of nodes and edges. Outputs of the input nodes are identical to the input parameters provided by the training samples and the edges deliver processed information from the *lower* nodes to the *higher* nodes amplifying by weight values. The hidden nodes and the output nodes receive processed and amplified information via incoming edges and execute activation for the sum of all receiving information.

The total number of nodes (N_{tot}) is the sum of the number of input nodes (N_{inp}),

hidden nodes (N_{hidden}), and output nodes (N_{out}). The number of input nodes is defined as identical to the number of input parameters in the training data set and the number of output nodes is defined as the number of output parameters in the training data set. The number of hidden units can vary from the given initial value as learning proceeds.

For the given number of hidden units, the network has the number of maximum possible feed-forward connections, $C_{forward}$, as a unique property of the network;

$$C_{forward} = N_{inp}(N_{hidden} + N_{out}) + N_{hidden}N_{out} + \frac{1}{2}N_{hidden}(N_{hidden} - 1) \tag{35}$$

Each and every node and edge is defined as follows.

4.1.2 Definition of Nodes

Each node i is defined by the following three properties;

1. The position index, i itself in this case, in the network as an integer value among 1 to N_{tot} where N_{tot} is the total number of nodes in the network; assuming feed-forward connections only, the inputs from the lower-index nodes only are allowed to every node.

2. The bias, b_i; every node has a bias parameter except the input nodes.

3. The activation function, g_i; such as one of linear, logistic sigmoid, and tangent sigmoid function types.

In the current study, the activation function for each and every node is assigned as one of the following three functions;

1. Linear activation function

$$g_i(a_i) = a_i \tag{36}$$

2. Tangent sigmoid activation function

$$g_i(a_i) = tanh(a_i) = \frac{e^{a_i} - e^{-a_i}}{e^{a_i} + e^{-a_i}} \tag{37}$$

3. Logistic sigmoid activation function

$$g_i(a_i) = \frac{1}{1 + e^{-a_i}} \tag{38}$$

For example, in the regression analysis case, the input and output nodes have linear activator, the hidden nodes have tangent sigmoid activator. In case of the classification problem, the input nodes have linear activator, the hidden nodes have tangent sigmoid activator, and the output nodes can be assigned as tangent sigmoid or logistic sigmoid activator depending on the non-dimensionalization of the training output data, i.e., if the training output data has been transformed to $[-1, 1]$ range, the tangent sigmoid is more suitable and, if the training output data has range of $[0, 1]$, the logistic sigmoid is more appropriate.

4.1.3 Definition of Edges

Each edge connecting node i and node j is defined by the following four properties in case of $i < j$;

1. The index of lower node, i
2. The index of higher node, j
3. The weight, w_{ji}
4. The optimization flag, o_{ji}; *false* for the *frozen* weight and *true* for the adjustable weight

4.1.4 Forward Propagation of Input Signals

At each training epoch, for the n-th training sample, each hidden or output node j has the output value of

$$z_j^{(n)} = g_j(\sum_{i=1}^{j-1}(w_{ji}z_i^{(n)}) + b_j), j > N_{inp}. \tag{39}$$

Here, any non-zero, feed-forward connection is allowed as a valid connectivity via its weight term while every zero-valued weight does not affect the calculation. In case of the input nodes, the node output values are assigned as the same as the input parameters of the particular training example n.

$$z_i^{(n)} = g_i(x_i^{(n)}), 1 \leq i \leq N_{inp}. \tag{40}$$

where $x_i^{(n)}$ is the i-th input parameter in the n-th training sample.

4.1.5 Backward Propagation of Error

At each training epoch, the training error for the n-th sample is

$$E^{(n)} = \frac{1}{2}\sum_{i=1}^{N_{out}}(z_{N_{inp}+N_{hidden}+i}^{(n)} - y_i^{(n)})^2 \tag{41}$$

where $y_i^{(n)}$ is the i-th target output in the n-th training sample. Because the effect of any weight change is transferred to the final output nodes via the sum of the information received in the destination node of corresponding connection,

$$a_j^{(n)} = \sum_{i=1}^{j-1}(w_{ji}z_i^{(n)}) + b_j, \tag{42}$$

it is convenient to apply the chain rule for this term to obtain the derivative of error with respect to the weight change as

$$\frac{\partial E^{(n)}}{\partial w_{ji}} = \frac{\partial E^{(n)}}{\partial a_j^{(n)}}\frac{\partial a_j^{(n)}}{\partial w_{ji}}. \tag{43}$$

Letting

$$\delta_j^{(n)} \equiv \frac{\partial E^{(n)}}{\partial a_j^{(n)}}, \tag{44}$$

the above equation becomes

$$\frac{\partial E^{(n)}}{\partial w_{ji}} = \delta_j^{(n)} \frac{\partial(\sum_{k=1}^{j-1}(w_{jk} z_k^{(n)}) + b_j)}{\partial w_{ji}} = \delta_j^{(n)} z_i^{(n)}. \tag{45}$$

Corresponding derivative of error with respect to the bias is

$$\frac{\partial E^{(n)}}{\partial b_j} = \delta_j^{(n)} \frac{\partial(\sum_{k=1}^{j-1}(w_{jk} z_k^{(n)}) + b_j)}{\partial b_j} = \delta_j^{(n)}. \tag{46}$$

In the above equations $\delta_j^{(n)}$ terms have to be calculated differently for the output nodes and for the hidden nodes. For the output nodes,

$$\begin{aligned} \delta_k^{(n)} &\equiv \frac{\partial E^{(n)}}{\partial a_k^{(n)}} \\ &= \frac{\partial E^{(n)}}{\partial z_k^{(n)}} \frac{\partial z_k^{(n)}}{\partial a_k^{(n)}} = \frac{\partial E^{(n)}}{\partial z_k^{(n)}} \frac{\partial g_k(a_k^{(n)})}{\partial a_k^{(n)}} \\ &= (z_k^{(n)} - y_k^{(n)}) g_k'(a_k^{(n)}) \end{aligned} \tag{47}$$

and, for the hidden nodes,

$$\begin{aligned} \delta_j^{(n)} &\equiv \frac{\partial E^{(n)}}{\partial a_j^{(n)}} \\ &= \sum_{k=j+1}^{N_{tot}} \frac{\partial E^{(n)}}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial a_j^{(n)}} = \sum_{k=j+1}^{N_{tot}} \delta_k^{(n)} \frac{\partial(\sum_{m=1}^{k-1} w_{km} g_m(a_m^{(n)}) + b_k)}{\partial a_j^{(n)}} \\ &= g_j'(a_j^{(n)}) \sum_{k=j+1}^{N_{tot}} \delta_k^{(n)} w_{kj}. \end{aligned} \tag{48}$$

Finally, the derivatives of total training error with respect to the weight and bias changes are, respectively,

$$\frac{\partial E}{\partial w_{ji}} = \sum_{n=1}^{P} \frac{\partial E^{(n)}}{\partial w_{ji}} = \sum_{n=1}^{P} z_i^{(n)} \delta_j^{(n)} \tag{49}$$

$$\frac{\partial E}{\partial b_j} = \sum_{n=1}^{P} \frac{\partial E^{(n)}}{\partial b_j} = \sum_{n=1}^{P} \delta_j^{(n)} \tag{50}$$

where P is the total number of example patterns in the training data set. The above derivation does not depend on the existence of established connection between two nodes and the effect of any connectivity exerts to the forward and backward information propagation depending only on its value of the weight allowing any non-zero, feed-forward connectivity participate as an effective network component.

4.1.6 Connectivity Adjustment

The weight and bias values are updated by the operation of optimization algorithm exploiting the derivatives of error calculated by the logic described in the previous section. This *unit* process is defined as an *epoch* which requires a single full sweep of whole training examples to collect information on the current network state. In the Optimal Brain Growth, the adjustment of connectivity is executed in the following steps assuming that there are $C_{forward}$ possible feed-forward connections and among them only C_{opt} weights are currently used as the adjustable parameters. As defined in the previous section, only those optimizable weights have $o_{ji} = \text{true}$ property while the other weights are designated as $o_{ji} = \text{false}$;

1. Find the connection whose error gradient is the minimum in its magnitude;

$$(i,j)_{min} = \underset{(i,j),o_{ji}=true}{\text{argmin}} \left| \frac{\partial E}{\partial w_{ji}} \right| \tag{51}$$

2. Find the *hidden* connection whose error gradient is the maximum in its magnitude;

$$(i,j)_{max} = \underset{(i,j),o_{ji}=false}{\text{argmax}} \left| \frac{\partial E}{\partial w_{ji}} \right| \tag{52}$$

3. Reset the optimization flag for selected connections;

$$o_{ji_{min}} \equiv o_{(i,j)_{min}} = false \tag{53}$$

$$o_{ji_{max}} \equiv o_{(i,j)_{max}} = true \tag{54}$$

4.1.7 Detecting Stabilized Network Component

Assuming the previously defined algorithm as a baseline mechanism for connectivity adjustment, several variants have been devised. One of them is to control the growth of a network by pruning the *stabilized* network components before the establishment of the new ones.

4.1.7.1 Stabilized Connection

A connection is regarded as a *stabilized* one ($s_{w_{ji}} = true$) when the variation of its weight value is very small under the pre-defined numerical threshold for finite number of training epochs consecutively;

$$N_{stag_{ji}}^{(n)} = \begin{cases} N_{stag_{ji}}^{(n-1)} + 1 & \text{if } \left| w_{ji}^{(n)} - w_{ji}^{(n-1)} \right| < \left| w_{ji}^{(n-1)} \right| \cdot r_{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{55}$$

$$s_{w_{ji}}^{(n)} = \begin{cases} true & \text{if } N_{stag_{ji}}^{(n)} > N_{stag_{max}} \\ false & \text{otherwise} \end{cases} \tag{56}$$

where n and $n-1$ indicate the current and the previous training epochs and the numerical thresholds such as $N_{stag_{max}}$ and $r_{threshold}$ have to be pre-assigned by network designer, by which the growth speed of a network is controlled.

4.1.7.2 Stabilized Neuron

A stability of a computational unit is defined similarly as the stability of a connection, but averaged values for an output of a neuron is used over the entire training examples.

$$N_{stag_j}^{(n)} = \begin{cases} N_{stag_j}^{(n-1)} + 1 & \text{if } \overline{(\delta z)_j^n} < r_{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{57}$$

where

$$\overline{(\delta z)_j^n} = \frac{\sum_{m=1}^{M} \left| \frac{z_{jm}^n - z_{jm}^{n-1}}{z_{jm}^{n-1}} \right|}{M} \tag{58}$$

where m indicates the m-th training example among total M examples and

$$s_{z_j}^{(n)} = \begin{cases} true & \text{if } N_{stag_j}^{(n)} > N_{stag_{max}} \\ false & \text{otherwise} \end{cases} . \tag{59}$$

4.2 Rearrangement for Levenberg-Marquardt Algorithm

While, in the standard BP algorithm, the derivatives of error is calculated directly for the squared-error term summed for all output nodes such as

$$E^{(n)} = \frac{1}{2}\sum_{i=1}^{N_{out}}(z^{(n)}_{N_{inp}+N_{hidden}+i} - y^{(n)}_i)^2, \tag{60}$$

in the LM algorithm, the Jacobian matrix has components of the derivative of error which is the residual error and the error needs to be individually defined for each of output nodes;

$$\epsilon^{(n,m)} = z^{(n)}_{N_{inp}+N_{hidden}+m} - y^{(n)}_m, 1 \le m \le N_{out} \tag{61}$$

where n indicates the n-th training sample and m the m-th output node. To evaluate the derivatives of the residual error term, δ has to be recalculated. For the output nodes,

$$\hat{\delta}^{(n,m)}_k \equiv \frac{\partial \epsilon^{(n,m)}}{\partial a^{(n)}_k} = \begin{cases} \frac{\partial \epsilon^{(n,m)}}{\partial z^{(n)}_k}\frac{\partial z^{(n)}_k}{\partial a^{(n)}_k} = \frac{\partial \epsilon^{(n,m)}}{\partial z^{(n)}_k}\frac{\partial g_k(a^{(n)}_k)}{\partial a^{(n)}_k} = g'_k(a^{(n)}_k) & \text{if} \quad k = m \\ 0 & \text{if} \quad k \neq m \end{cases} \tag{62}$$

and, for the hidden nodes,

$$\begin{aligned} \hat{\delta}^{(n,m)}_j &\equiv \frac{\partial \epsilon^{(n,m)}}{\partial a^{(n)}_j} \\ &= \sum_{k=j+1}^{N_{tot}} \frac{\partial \epsilon^{(n,m)}}{\partial a^{(n)}_k}\frac{\partial a^{(n)}_k}{\partial a^{(n)}_j} = \sum_{k=j+1}^{N_{tot}} \hat{\delta}^{(n,m)}_k \frac{\partial(\sum_{m=1}^{k-1} w_{km} g_m(a^{(n)}_m) + b_k)}{\partial a^{(n)}_j} \\ &= g'_j(a^{(n)}_j)\sum_{k=j+1}^{N_{tot}} \hat{\delta}^{(n,m)}_k w_{kj}. \end{aligned} \tag{63}$$

The resultant derivatives of residual error for the m-th output value with respect to the changes of weight and bias for the n-th training sample are, respectively,

$$\frac{\partial \epsilon^{(n,m)}}{\partial w_{ji}} = z^{(n)}_i \hat{\delta}^{(n,m)}_j \tag{64}$$

$$\frac{\partial \epsilon^{(n,m)}}{\partial b_j} = \hat{\delta}^{(n,m)}_j \tag{65}$$

The Jacobian matrix $\mathbf{Z}$ can be arranged in the following structure following the work described by Wilamowski [92].

$$\mathbf{Z} = \begin{pmatrix} \frac{\partial \epsilon^{(1,1)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(1,1)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(1,1)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(1,1)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(1,1)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(1,1)}}{\partial b_{N_{tot}}} \\ \frac{\partial \epsilon^{(1,2)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(1,2)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(1,2)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(1,2)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(1,2)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(1,2)}}{\partial b_{N_{tot}}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \epsilon^{(1,M)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(1,M)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(1,M)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(1,M)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(1,M)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(1,M)}}{\partial b_{N_{tot}}} \\ \frac{\partial \epsilon^{(2,1)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(2,1)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(2,1)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(2,1)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(2,1)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(2,1)}}{\partial b_{N_{tot}}} \\ \frac{\partial \epsilon^{(2,2)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(2,2)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(2,2)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(2,2)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(2,2)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(2,2)}}{\partial b_{N_{tot}}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \epsilon^{(2,M)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(2,M)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(2,M)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(2,M)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(2,M)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(2,M)}}{\partial b_{N_{tot}}} \\ \frac{\partial \epsilon^{(3,1)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(3,1)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(3,1)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(3,1)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(3,1)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(3,1)}}{\partial b_{N_{tot}}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \epsilon^{(P-1,M)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(P-1,M)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(P-1,M)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(P-1,M)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(P-1,M)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(P-1,M)}}{\partial b_{N_{tot}}} \\ \frac{\partial \epsilon^{(P,1)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(P,1)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(P,1)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(P,1)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(P,1)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(P,1)}}{\partial b_{N_{tot}}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \epsilon^{(P,M)}}{\partial w_{(1)}} & \frac{\partial \epsilon^{(P,M)}}{\partial w_{(2)}} & \cdots & \frac{\partial \epsilon^{(P,M)}}{\partial w_{(C_{opt})}} & \frac{\partial \epsilon^{(P,M)}}{\partial b_{N_{inp}+1}} & \frac{\partial \epsilon^{(P,M)}}{\partial b_{N_{inp}+2}} & \cdots & \frac{\partial \epsilon^{(P,M)}}{\partial b_{N_{tot}}} \end{pmatrix} \tag{66}$$

where P is the total number of data samples in the training set, M is the number of output parameters, and the weights are re-indexed by unique identity only for optimizable ones excluding *frozen* weights. The residul error vector is defined in a straightforward way such as

$$\boldsymbol{\epsilon}^T = \begin{pmatrix} \epsilon^{(1,1)} & \epsilon^{(1,2)} & \ldots & \epsilon^{(1,M)} & \epsilon^{(2,1)} & \ldots & \epsilon^{(P-1,M)} & \epsilon^{(P,1)} & \ldots & \epsilon^{(P,M)} \end{pmatrix} \tag{67}$$

Also the vector of adjustable parameters has the following form;

$$\mathbf{W}^T = \begin{pmatrix} w_{(1)} & w_{(2)} & \ldots & w_{C_{opt}} & b_{N_{inp}+1} & b_{N_{inp}+2} & \ldots & b_{N_{tot}} \end{pmatrix} \tag{68}$$

Using the constructed matrices, the new weight and bias values are calculated by following equation advancing to the next iteration;

$$\mathbf{W_{new}} = \mathbf{W_{old}} - (\mathbf{Z^T Z} + \lambda \mathbf{I})^{-1} \mathbf{Z^T} \boldsymbol{\epsilon}(\mathbf{W_{old}}) \tag{69}$$

where $\mathbf{I}$ is the identity matrix and λ is the step size for the iteration.

4.3 Addition of a New Computational Unit

In the Optimal Brain Growth, following steps are executed when an additional node has to be introduced;

1. Find the output node whose sum of squared-error is the maximum among all output nodes;

$$m_{max} = \underset{m,1\leq m\leq N_{out}}{\operatorname{argmax}} \sum_{n=1}^{P} \left(z^{(n)}_{N_{inp}+N_{hidden}+m} - y^{(n)}_{m} \right)^2 \tag{70}$$

2. Calculate the correlation of each and every node's output value and the residual error of the maximum error output which has been selected in the previous step;

$$S_i = \left| \sum_{n=1}^{P} (z_i^{(n)} - \bar{z}_i)(e^{(n)}_{m_{max}} - \bar{e}_{m_{max}}) \right| \tag{71}$$

where

$$e^{(n)}_{m_{max}} = z^{(n)}_{N_{inp}+N_{hidden}+m_{max}} - y^{(n)}_{m_{max}} \tag{72}$$

and $\bar{z}_i$ and $\bar{e}_{m_{max}}$ is the mean value for the whole training examples.

3. Select the node whose correlation value is the maximum and establish two adjustable connections between selected nodes and the new node whose node index, j, has to be determined as the larger than the maximum correlation node's and the smaller than the minimum index of the output nodes;

$$i_{max} = \underset{i,1\leq i\leq N_{inp}+N_{hidden}}{\operatorname{argmax}} S_i \tag{73}$$

$$j = \text{random integer}, i_{max} < j \leq N_{inp} + N_{hidden} \tag{74}$$

4. Re-index all the nodes whose node index is equal to or greater than j making room for a new node and update all the connection definitions responding to

this event and then,

$$o_{ji_{max}} = true \quad (75)$$

$$o_{(N_{inp}+N_{hidden}+m_{max})j} = true \quad (76)$$

5. Assign weight and bias values for the newly added components;

$$w_{ji_{max}} = \text{random number between } -0.5 \text{ and } 0.5 \quad (77)$$

$$b_j = \text{random number between } -0.5 \text{ and } 0.5 \quad (78)$$

$$w_{(N_{inp}+N_{hidden}+m_{max})j} = 0 \quad (79)$$

4.4 Decomposition of an Existing Neuron

In the Optimal Brain Growth, following steps are executed when the an existing neuron has to be decomposed;

1. Detect a stabilized neuron

2. Decompose that neuron into two neurons by following rules;

   - Two decomposed neurons have identical input connections to the orgininal neuron's in their connectivity and weight values
   - Two decomposed neurons have identical bias values to the original neuron's bias
   - Two decomposed neurons have identical output connections to the original neuron's in their connectivity only and have randomized weight values whose sum for the same output neuron is identical to the original neuron's connection

3. Re-index all the nodes whose node index is equal to or greater than the decomposed neurons and update all the connection definitions responding to this event

4.5 Summary of Procedure

The overall implementation steps of the Optimal Brain Growth algorithm can be summarized as follows;

1. Setup a network with the given number of nodes and connections, compatible with the number of input and output parameters of the training examples and depending on the choices for number of hidden nodes and the minimal connection structure.

2. Initialize all adjustable parameters with *small random* numbers.

3. Calculate the derivatives of error in the form of summed squared-error with respect to all possible connectivity.

4. If necessary, adjust network connectivity based on the result of previous step.

5. If necessary, add new network components.

6. Calculate the derivatives of residual error with respect to all adjustable parameters determined through Step $4-5$.

7. Iterate with the LM algorithm to calculate the new weight and bias set

8. Terminate training procedure or repeat the above process re-starting from the Step 3 depending on the convergence status.

Figure 28 shows the simplified flow charts for this procedure compared to the conventional weight-only training.

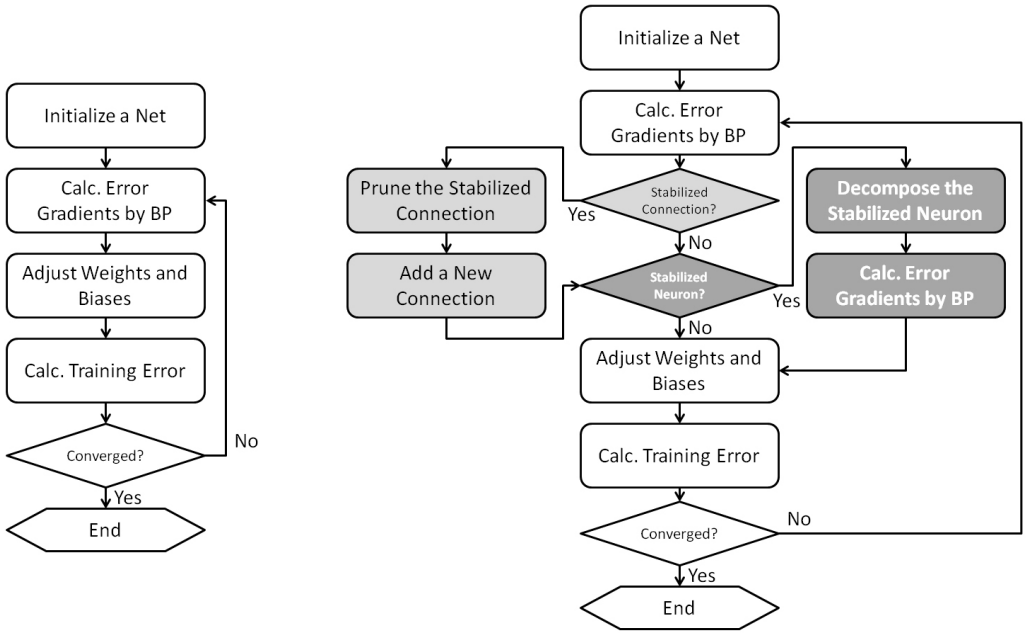


Figure 28: Conventional weight-only training (left) and the OBG training (right)

4.6 Numerical Implementation

The described OBG algorithm has been coded in the generic C++ language resulting in, approximately, less than $2,000$ programming lines. For the linear algebraic computation, the Armadillo, an open source C++ linear algebra library [75], has been used, which not only supports straightforward programming of the code but also provides relatively faster running speed of the code. Figure 29 shows the benefit in computation speed of the Armadillo library compared to other contemporary software.

Operation	Matlab	Octave	Newmat	IT++
Addition and scalar multiplication	2.9	4.4	2.8	3.9
Transpose, matrix multiplication and in-place addition	1.0	1.3	4.0	1.1
Decreasing size matrix multiplication	1.8	2.1	12.5	2.0
Submatrix copy	3.6	11.5	2.6	10.6
Direct element access	14.7	2592.3	2.5	3.1

Table 10: Speedup factor of Armadillo relative to other software, using in-cache matrices. Tests were performed on an Intel Core2 Duo CPU with 2 Mb cache, running in 64 bit mode at 2 GHz. Each matrix element was stored as a double precision floating point number (8 bytes). A Linux based operating system was used (Fedora 12), incorporating Linux kernel v2.6.32 and the GCC v4.4.4 C++ compiler. Versions of software were as follows: Armadillo 0.9.80, Matlab v7.1.0.183 SP3 64-bit, Octave v3.2.3, Newmat 11 beta, IT++ v4.0.6.

Figure 29: Speedup factors of Armadillo [75]

CHAPTER V

EXPERIMENTS

Based on the need for an integrated learning mechanism both for connectivity as well as for weights of the neural network, the Optimal Brain Growth algorithm has been formulated. But as a novel learning algorithm, its basic characteristics on the practical training problems have to be investigated and the activation criteria for the growth of connections and computational units have to be streamlined for the successful application. One of the most important issues is the vulnerability to the over-training or over-fitting problem as a constructive method. In this chapter, several numerical experiments are described in the comparative manner with the conventional MLP networks for their training efficiency, generalization performance, and the robustness to the random initial conditions.

5.1 First Experiment

To probe the basic training performance of the OBG, a simple fluid mechanics problem has been modeled. This particular training set has four input parameters; pressure difference between two end points of the cylindrical pipe, opening ratio of inside valve (0.0 to 1.0) which is located at the middle of the pipe, length of the pipe, and diameter of the pipe. The single output response is the volumetric flow rate per unit time. Although there exist many direct ways to obtain the solution for this elementary fluid mechanics problem via numerical methods, the surrogate modeling is very effective mean when confronted a massive fluid network consisting of hundreds to thousands of unit pipe such as in the infrastructure design and the ship design tasks [59]. Total 522 samples had been obtained by physics-based analysis using commercial software FlowMaster [18]. To build a surrogate model for this problem, all of the 522 samples

have been used for the network training without monitoring validation error. For the OBG, the growth rate of the connectivity is one per each epoch and the addition of the new computational units has not been applied in this experiment. The original training sample data are non-dimensionalized to the values between -1.0 and 1.0. The input and output nodes in the network have the linear activation function and the hidden nodes have the tangent sigmoid activation function.

5.1.1 Learning Capability of the OBG

In the first set of network trainings, the introduction of new hidden units has not been allowed and the magnitude-based, hard pruning has been applied to maintain the same numbers of network components, i.e., the number of neurons and the number of connections. Figure 30 shows the convergence trajectory in terms of Mean Squared Error (MSE) for the training examples. Starting from a randomly initialized single-hidden layer MLP, the OBG training converges to the significantly lower MSE than the conventional weight-only training. Here, the *convergence* is defined as the network status having very low variation in the training MSE during the finite, consecutive training epochs. The initial and the converged network configurations are depicted in Figure 31 for the case of the OBG training, in which the restructuring of the connectivity from the MLP is clearly observed. An important and, in the same time, interesting question is whether this change in the network connectivity is a genuine *adaptation* to the given training examples in the similar token for the weights to be said as *adapted* by the weight-only training. Considering the uncertainty in the random initial condition, this single result is hard to be generalized to draw any conclusion. Hence, we used a group of 50 comparative cases to repeat this type of experiments over more diverse network sizes (8, 10, 12, 14, 16 hidden units) and initial conditions (10 random trials for each number of hidden unit). Figure 32 summarizes the final, converged MSEs confirming the general trend of improved

learning capability of the OBG training for the currently given problem. In all training cases, the difference in time consumption per training epoch was negligible between two training methods.

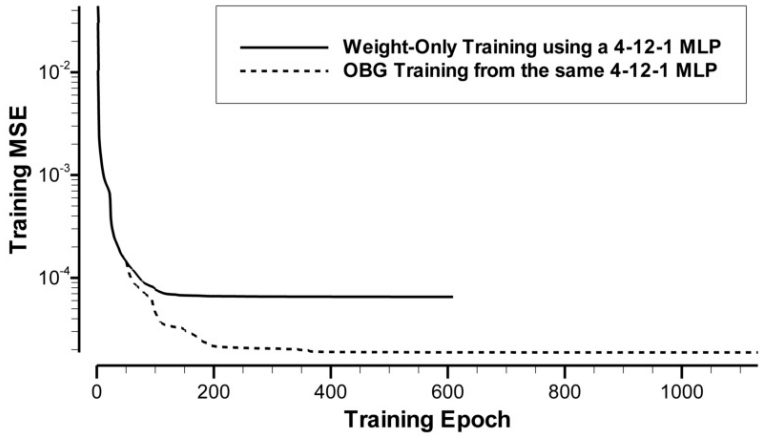


Figure 30: An example of training error variation during the weight-only training and the OBG training (no additional neurons allowed)

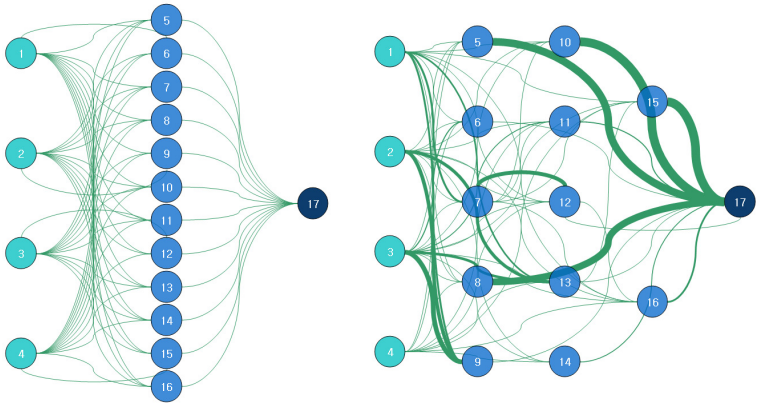


Figure 31: The initial 4-12-1 MLP network (left) and the OBG training result (right, the thickness of connection represents the absolute magnitude of weight.)

In the next set of network trainings, the impact of neuronal decomposition has been observed allowing the finite number, 4, of hidden units to be added unless the network converges to local minimum earlier than the activation of the decomposition mechanism (or earlier than the occurrences of the stabilized hidden units). As shown in Figure 33, the OBG training automatically, or following the simple rule of detecting

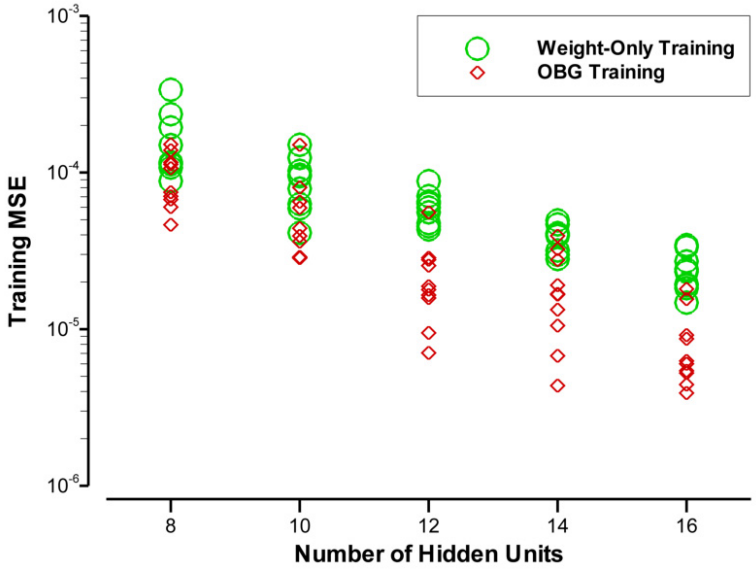


Figure 32: Minimum training errors obtained from the weight-only training and the OBG training (no additional neurons allowed

stabilized hidden unit, introduces new hidden units and, exploiting this feature, converges to the significantly lower MSE by orders-of-magnitude difference. Moreover, this enhanced learning capability by the neuronal decomposition has the potential to be beyond (or to be synergetic with) the advantage of the increased network size, i.e., many cases of the OBG training with initial N hidden units (allowed to have +4 hidden units during the training) outperformed the weight-only trainings with initial $N + 4$ hidden units (Figure 30). As an observation from the current experiment, at least for a certain type of supervised learning task, the connectivity adjusting learning scheme has advantage over the conventional weight-only learning scheme by adapting the connectivity as well as the weights for the given training examples.

5.1.2 Comparison with Double-Hidden Layer MLP

Considering the uncertainty originating from the random initial conditions and the termination criteria for training process, a series of training campaigns has been executed;

- 3 training error goals; 10^{-4}, 10^{-5}, and 10^{-6} of the half value of the mean squared

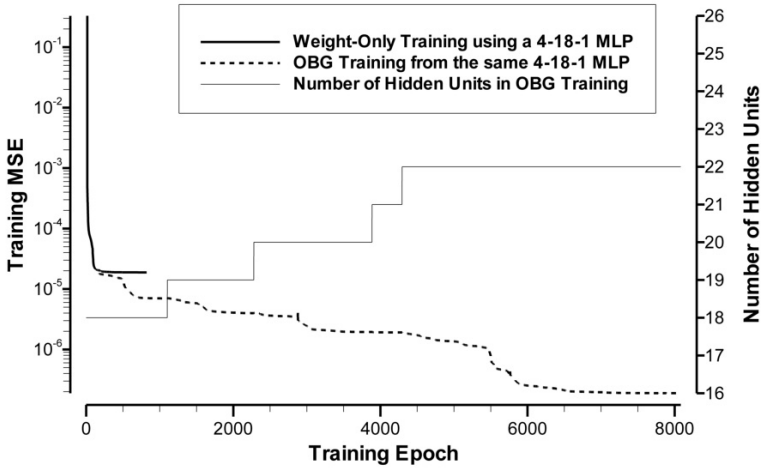


Figure 33: An example of training error variation during the weight-only training and the OBG training (4 additional neurons allowed)

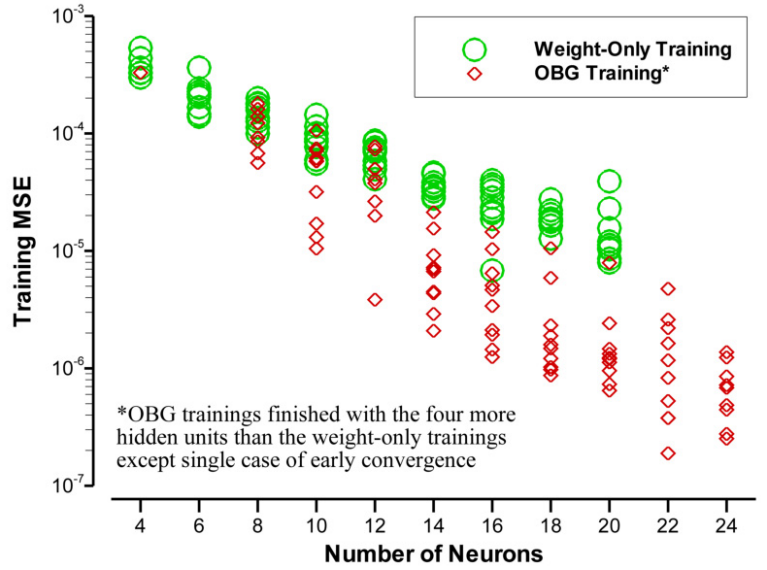


Figure 34: Minimum training errors obtained from the weight-only training and the OBG training (4 additional neurons allowed

error have been used as the training termination criteria under the constraint of maximum training epochs of $5,000$.

- 7 initial network structures; two-layer MLP structure with 4-2-2-1 (4 input nodes, 2 hidden nodes in the first hidden layer, 2 hidden nodes in the second hidden layer, 1 output node), 4-3-3-1, 4-4-4-1, 4-5-5-1, 4-6-6-1, 4-7-7-1, 4-8-8-1 layouts have been used as the initial network.

- Each and every network has been repeatedly trained from 10 random initial parameter sets for its initial weights and biases.

Therefore, total $210(3 \times 7 \times 10)$ training cases have been executed both for the MLP training and the OBG training. The MLP training consists of the standard BP and LM method. Each comparative pair of the MLP and the OBG training has been initiated from the exactly identical initial network structure and weight/bias values. One example from the result is shown in Figure 35. Here, the convention for the network diagrams is;

- The color of the nodes discriminates input nodes, hidden nodes, and output nodes and each node has its identity number.

- The color of the connections indicates its sign, i.e., orange for positive weight connection and green for the negative weight connection.

- The solid line connections have *adjustable* weight and the dashed line means *frozen* connection.

As clearly seen in Figure 35, the OBG adjusts connectivity as well as weight values resulting in a dissimilar structure contrast to the MLP trained network which has been changed in its weight values. In this particular case, the OBG trained initial 4-3-3-1 network faster by altering its connection structure than MLP which has the fixed connectivity between the nodes.

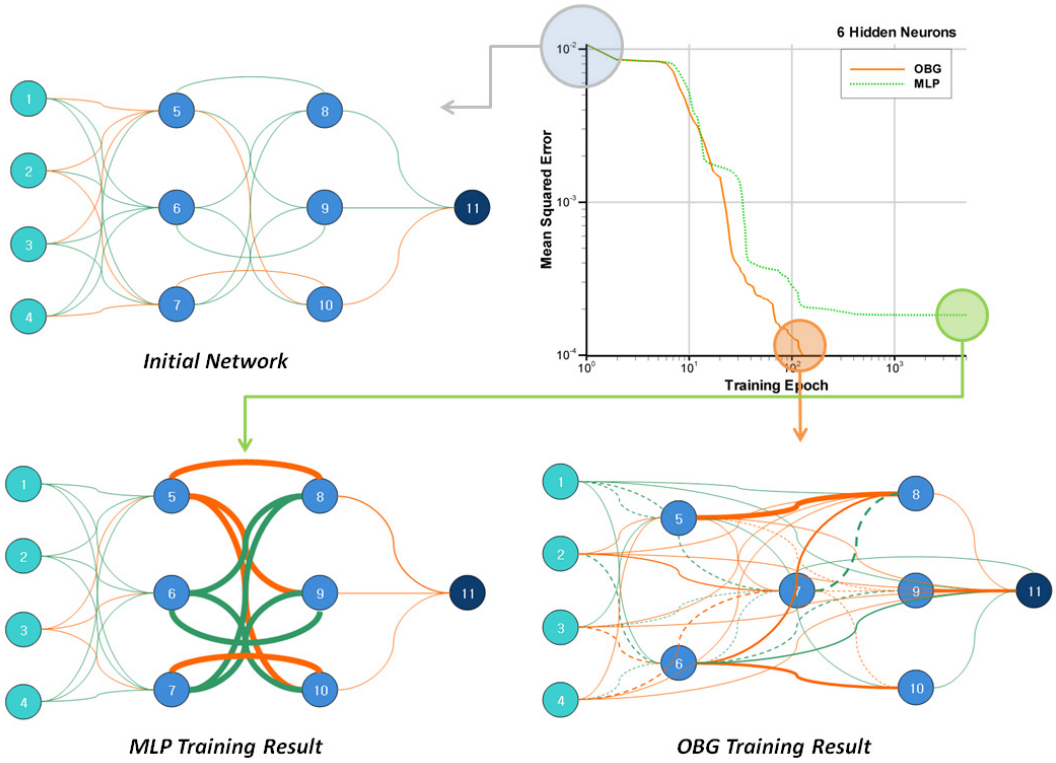


Figure 35: One example from the results of Experiment 1

Table 1 shows the convergence status of all 210 training cases and Figure 36-42 contain the variation of training error for all cases. By comparing the MLP and the OBG training results both of which begin with the same initial network states, following characteristics are observed;

- For the given error goal, overall convergence rate is much higher in case of the OBG training, especially, when the smaller number of hidden nodes are used.
- The dissimilarity between two approaches in the pattern of training error variation is much larger when the small number of hidden nodes are used. The occurrence of *plateau* region in the error variation indicating stagnated error reduction is clearly reduced in the OBG training.

Table 1: Result of Experiment 1

Error Goal	Initial Net	Convergence Rate (*)	
		MLP	OBG
10^{-4}	4-2-2-1	0/10 (N/A)	3/10 (265)
	4-3-3-1	9/10 (171)	8/10 (155)
	4-4-4-1	10/10 (85)	10/10 (104)
	4-5-5-1	10/10 (74)	10/10 (72)
	4-6-6-1	10/10 (74)	10/10 (71)
	4-7-7-1	10/10 (62)	10/10 (62)
	4-8-8-1	10/10 (51)	10/10 (51)
10^{-5}	4-2-2-1	0/10 (N/A)	0/10 (N/A)
	4-3-3-1	0/10 (N/A)	2/10 (541)
	4-4-4-1	0/10 (N/A)	8/10 (750)
	4-5-5-1	3/10 (201)	10/10 (480)
	4-6-6-1	10/10 (211)	10/10 (201)
	4-7-7-1	10/10 (143)	10/10 (211)
	4-8-8-1	10/10 (131)	10/10 (139)
10^{-6}	4-2-2-1	0/10 (N/A)	0/10 (N/A)
	4-3-3-1	0/10 (N/A)	0/10 (N/A)
	4-4-4-1	0/10 (N/A)	0/10 (N/A)
	4-5-5-1	0/10 (N/A)	0/10 (N/A)
	4-6-6-1	6/10 (1209)	5/10 (1119)
	4-7-7-1	8/10 (469)	10/10 (641)
	4-8-8-1	9/10 (266)	10/10 (359)

*Average number of epochs to reach error goal

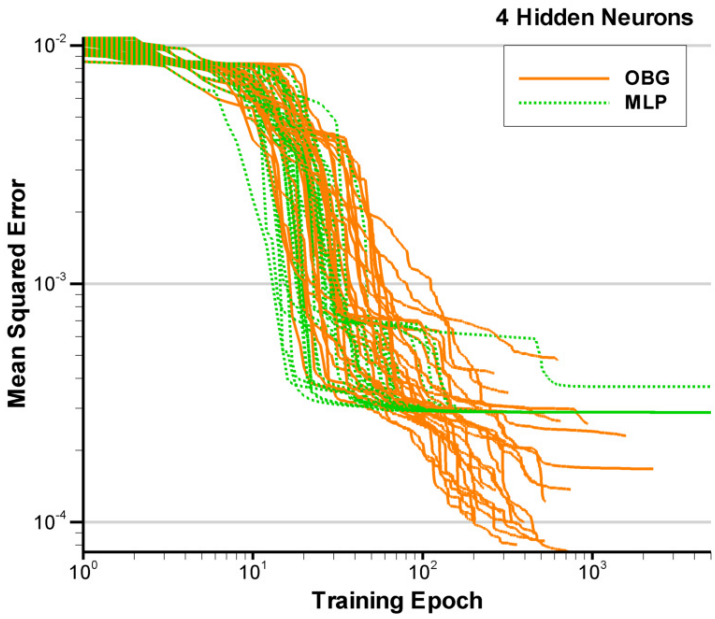


Figure 36: Training error history, 4 hidden neurons, Experiment 1

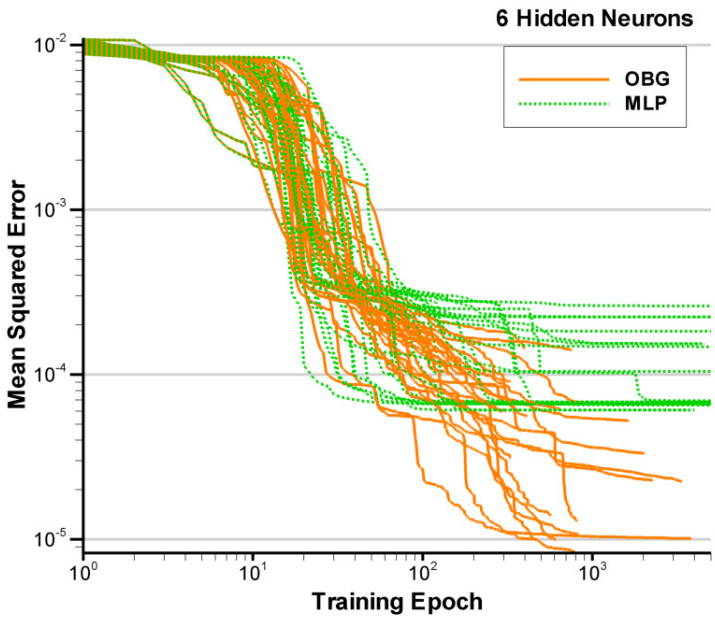


Figure 37: Training error history, 6 hidden neurons, Experiment 1

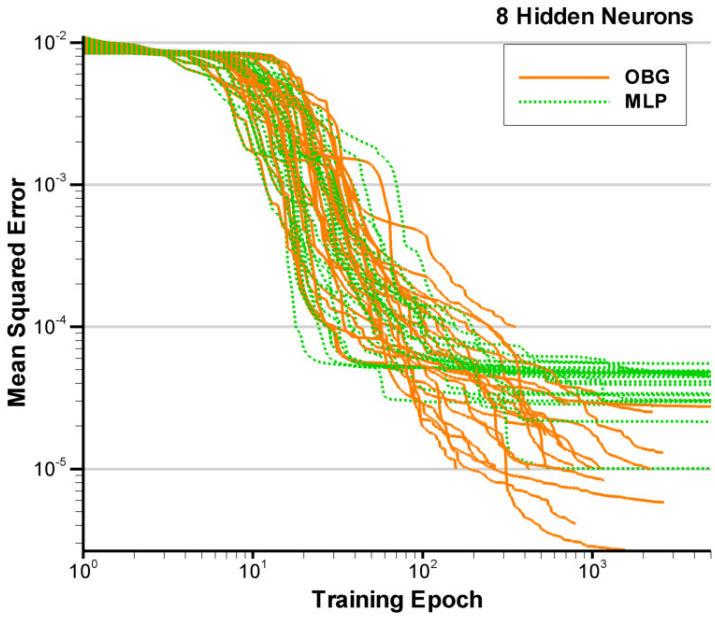


Figure 38: Training error history, 8 hidden neurons, Experiment 1

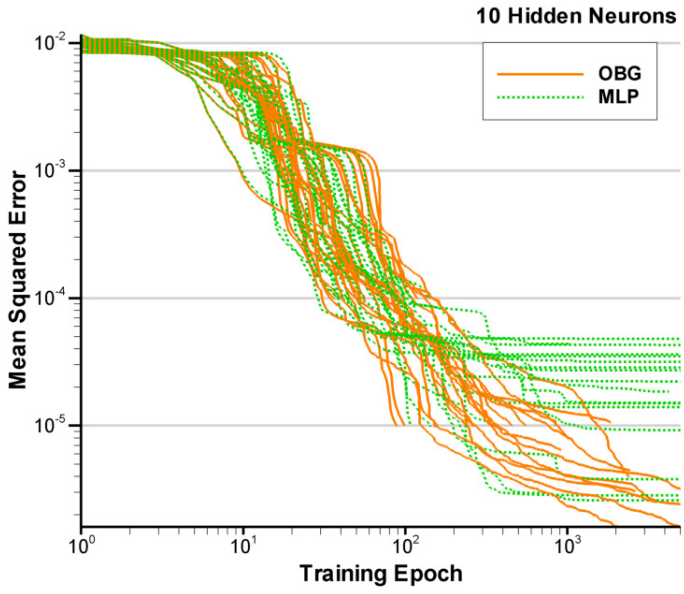


Figure 39: Training error history, 10 hidden neurons, Experiment 1

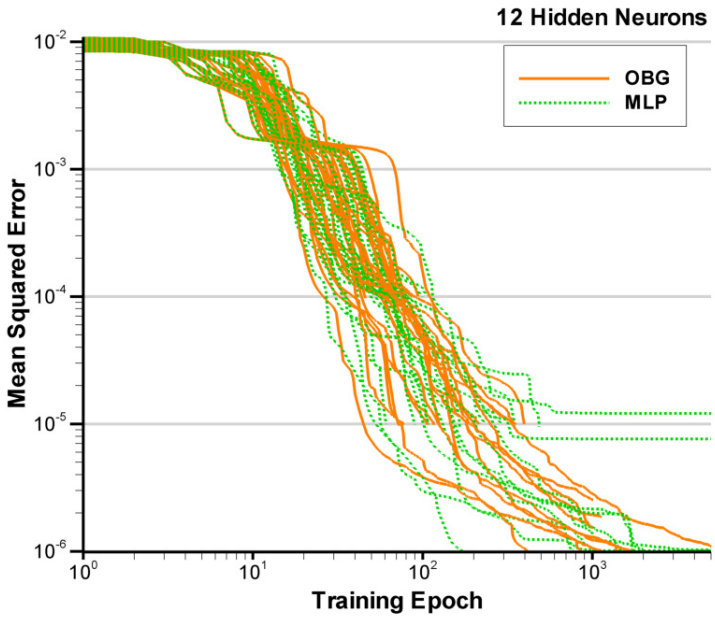


Figure 40: Training error history, 12 hidden neurons, Experiment 1

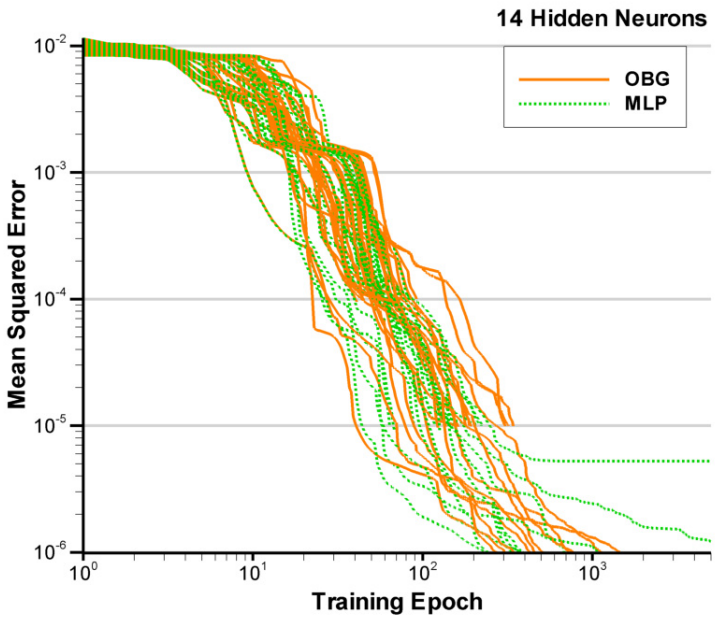


Figure 41: Training error history, 14 hidden neurons, Experiment 1

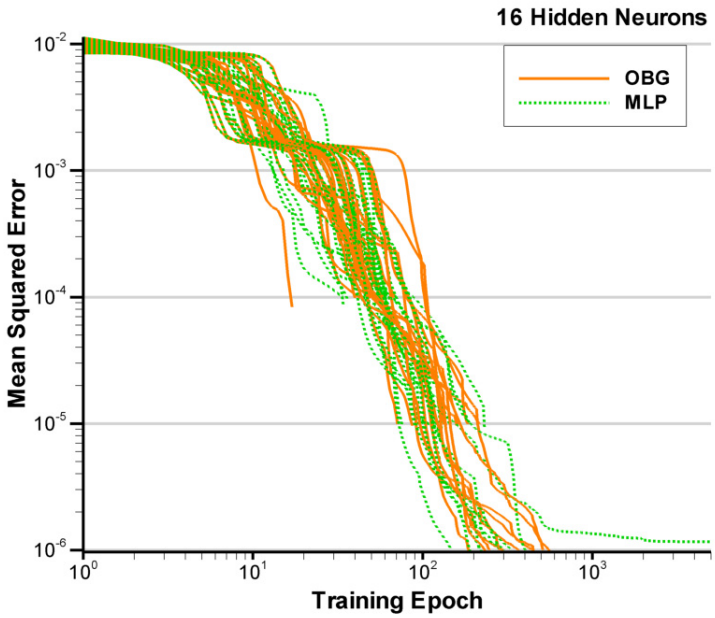


Figure 42: Training error history, 16 hidden neurons, Experiment 1

5.2 Second Experiment; Two-Spiral Classification Problem

As another benchmarking test for the training performance of the OBG, the Two-Spiral Problem has been chosen because the abundance of comparative studies for diverse network architectures have been reported using this problem in the literature. The task is for the neural networks to learn a mapping which distinguishes between points on two intertwined spirals as shown Figure 43.

Since the first introduction of Two-Spiral Problem by Wieland [47], it has been regarded as one of the most difficult problem to be solved by standard back propagation neural network. As a brief literature survey, the research activities to solve this problem with the ANN were as follows;

- Land and Witbrock reported that the solution could not be obtained with a standard back propagation neural network. Only with additional *short-cut* links extending the standard MLP structure to generalized MLP (GMLP) or bridged MLP (BMLP) network, they could obtain the solution within $20,000$ training

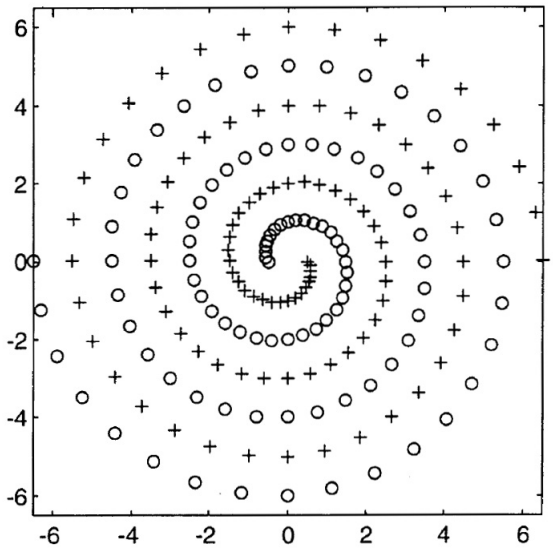


Figure 43: Two-Spiral Problem [1]

epochs using 2-5-5-5-1 network (three hidden layers of five units each) [47, 24].

- Baum and Land failed to a correct solution with 2-50-1 back propagation network starting from random initial weights [47].

- Fahlman and Lebiere demonstrated the capability of Cascade Correlation network by effectively solving this problem [24]

- Denoeux and Lengelle used a 2-20-1 network and obtained solution within $1,200$ training epochs. They additionally used extensive vector quantization to generate prototypes for initialization [47].

- Hwang et al. demonstrated the capability of Projection Pursuit learning network with the hermite polynomial type activation functions by obtaining much smoother solutions compared to the Cascade Correlation method's result [44].

- Wilamowski demonstrated the efficiency of the fully-connected cascade (FCC) network solving this problem by the second-order weight optimization compared to the standard error back propagation algorithm [90].

5.2.1 General Training Result

The MLP networks have been trained using conventional BP-LM method and the OBG method to solve Two-Spiral Problem. Total 14 initial network structures consisting of 7 single-hidden layer MLP networks and 7 double-hidden layer MLP networks have been used and each pair of MLP-OBG trainings from the same network structure has been repeated for 20 random initial parameter sets for their initial weights and biases. In this way, as in the previous experiment's case, each and every MLP-OBG training pair has exactly same initial network conditions. The training data samples have been non-dimensionalized to $[-1.0, 1.0]$ range and all nodes have tangent sigmoid activation function except the input nodes which have the linear activation function. For all training cases, the maximum limit in the number of iterations has been fixed with $5,000$.

Table 2 summarizes the training results from total 280 different initial network states. Overall, the chance to obtain a correctly classifying solution for all 194 data points is much higher in the OBG training results, especially, in cases of the initial structure of the single-hidden layer MLP. In cases of existence of comparable numbers of solutions for both training methods, the OBG requires less number of training epochs compared to the MLP training. Another comparison is given in Figure 45 which compares the number of incorrectly classified points out of total 194 training samples by the trained networks regardless of its complete success or not. This result tells that, within the given number of the maximum training epochs, the networks trained by OBG algorithm are consistently outperforming their baseline MLP networks and, moreover, the training results by OBG algorithm is far less sensitive to its initial structure while there exists significant difference in training error between single- and double-hidden layer MLP networks.

Another notable finding is that, contrast to the descriptions in the literature, the networks having typical double-hidden layer architecture, in case of using more than

14 hidden nodes, result in successful solutions without special treatment for their initial conditions although all of the single-hidden layer MLP networks failed to converge to correct solution (except 1 case out of 140 trials as shown in the Table 2). The possible explanation for this discrepancy in the difficulty of obtaining solution for the Two-Spiral Problem is that the success or failure to learn the correct solution in this case depends heavily on the training algorithm as well as the network architecture, i.e., the training results before the application of the significantly more efficient second-order optimization techniques such as the LM algorithm are not suitable to be directly compared with more recent results including the current work to analyze the advantage of a particular network architecture. One example showing the dramatic increase in the training efficiency using the common network FCC architecture is reported in the work of Wilamowski [90];

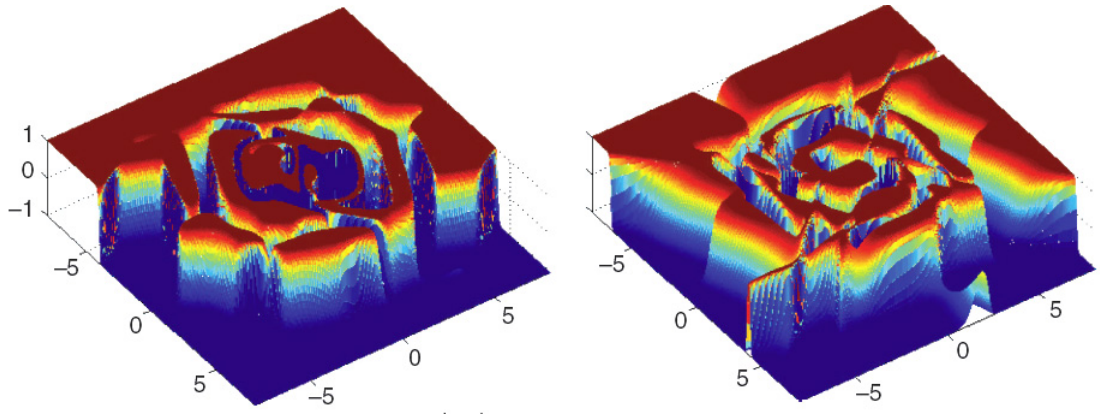


Figure 44: Solutions for Two-Spiral Problem using FCC networks [90]

Figure 44 shows two solutions using FCC networks. Left solution is obtained by NBN (neuron-by-neuron, [90]) algorithm which has similar level of training efficiency to the LM algorithm as the second-order optimizer and right solution is by error back-propagation with the gradient descent optimization. Although both solutions are correctly classifying all training samples for the Two-Spiral Problem, second-order optimization required only 244 iterations for 8-neuron, 52-weight network while the

Table 2: Result of training for Two-Spiral Problem

Initial Net	Misclassifications*		Solution Rate (**)	
	MLP	OBG	MLP	OBG
2-8-1	68.5/194	25.5/194	0/20 (N/A)	1/20 (2824)
2-10-1	59.7/194	18.1/194	0/20 (N/A)	2/20 (660)
2-12-1	43.0/194	12.0/194	0/20 (N/A)	7/20 (1821)
2-14-1	29.3/194	3.90/194	0/20 (N/A)	6/20 (724)
2-16-1	22.5/194	2.25/194	0/20 (N/A)	9/20 (1818)
2-18-1	12.3/194	1.40/194	0/20 (N/A)	10/20 (892)
2-20-1	7.50/194	1.35/194	1/20 (202)	18/20 (896)
2-4-4-1	48.6/194	27.2/194	0/20 (N/A)	1/20 (415)
2-5-5-1	33.5/194	16.5/194	0/20 (N/A)	3/20 (529)
2-6-6-1	14.1/194	4.45/194	1/20 (571)	4/20 (2749)
2-7-7-1	6.20/194	2.70/194	8/20 (1206)	12/20 (880)
2-8-8-1	4.30/194	0.45/194	11/20 (582)	16/20 (338)
2-9-9-1	3.05/194	0.15/194	14/20 (328)	17/20 (297)
2-10-10-1	0.20/194	0.05/194	18/20 (209)	19/20 (227)

*Average number of wrong classifications for 20 training sets
**Average number of epochs to reach the solution

first-order optimization required $308,325$ iterations for 16-neuron, 168-weight network. But the second-order optimizers not only provide faster convergence but also eliminate the uncertainties originated from the step size of the learning on which the training results of the standard BP training depend. Hence, the second-order weight optimization is more robust in this aspect. The faster and more robust convergence allows the more trials to obtain solutions affecting the general observation and conclusion on the difficulty of particular problems. Therefore, the fair comparison in the architectural level has to be performed using the same training framework such as in the current experiment where both the MLP networks and the OBG training have been performed using the LM algorithm for their weight optimization.

5.2.2 Detailed Comparison

Beyond the general trend of training performance, more detailed discussion on the learning behaviors of two training methods is given in this section. Among the initial network states from which the MLP and the OBG training have solved the problem

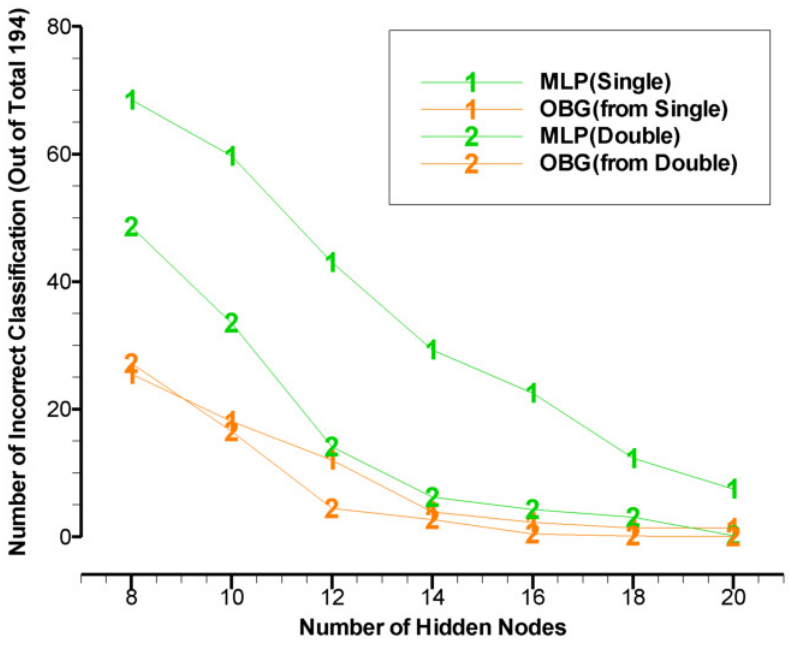


Figure 45: Training performance comparison, Two-Spiral Problem

correctly, three cases have been chosen to this purpose. Figure 47-49 show the training error history, initial and final converged networks, and the classification results for the training data samples using the final converged networks. In the network diagram, the following legends for connections are used, differently from the presentation of the previous experiment;

- Regardless of the sign of weight values, every connection is shown with solid line.
- Green connection is *frozen* connection whose weight has been excluded from the optimization process.
- Orange connection is *adjustable* connection whose weight is included as the optimization parameter.

The evident observations from these three cases are;

- The resultant networks by the MLP training have relatively larger magnitude in their weight values indicated by the thicker connections in the network diagrams.
- The OBG trainings end sooner not only by reducing error faster but also by satisfying the convergence criterion earlier even with the relatively larger error than in the MLP trainings.
- The solution has evidently smoother and more continuous classification boundaries in case of the OBG training.

First, the larger weight in its magnitude usually indicates the saturation in the neuron's activation, i.e., the neuron's output value is at near extremal one of the activation range such as the very close values to one of either -1.0 or 1.0 in case of the neuron having tangent sigmoid activation. Due to the limitation near the extremum of the activation range significantly reducing the impact of weight changes in the pre-node weight adjustments, the post-node weights grow larger in its magnitude to the error reducing direction. Early saturation of the neuron is known to have harmful effect on the proper regression performance [44]. Second, the pattern observed in the variation of error near the final convergence is also related to the first observation. The termination criterion to determine convergence is whether all the training examples are correctly discriminated or not. Here, correct discrimination is defined as the identical sign of the output node to the target, i.e., if the output node results in any positive real number for the target value of $+1$, then it is regarded as correct classification, which is a proper criterion only for the two-state (or binary) classification problem such as Two-Spiral Problem. Hence, two successful solutions may result in different residual error values. Logically, the current observation must be resulted from that more of the training examples fail to change to a proper sign in output node's output in case of the MLP training while the overall residual error is decreasing further during this *delay*. The observation on the early saturation

of neurons in the MLP training indicated by significantly more *heavy* weights may cause this problem because the saturation of the neuron can be thought of reduced adaptability of the network. Roughly speaking, the network in the OBG training has more chance to avoid saturation of neurons by changing adjustable connectivity toward higher gradient connections rather than being stagnated by the connections linked from the saturated neuron whose sensitivity to the input signal is infinitesimal and, hence, reducing error only by increasing magnitude of weight values. Finally, the resultant smoother and more continuous classification boundaries can be compared with those of other contemporary methods. Figure 46 shows the classification boundaries of the Cascade Correlation (left), Projection Pursuit using supersmoother (middle), and Projection Pursuit using hermite polynomial (right). As long as all training examples are correctly classified, any classification boundary is a satisfactory solution. But the smoother and more continuous boundary can be regarded as an indication of the better generalization performance. Consistently, the OBG result is smoother and more continuous than the MLP result corresponding to the identical initial network states and also comparable to the results shown in Figure 46.

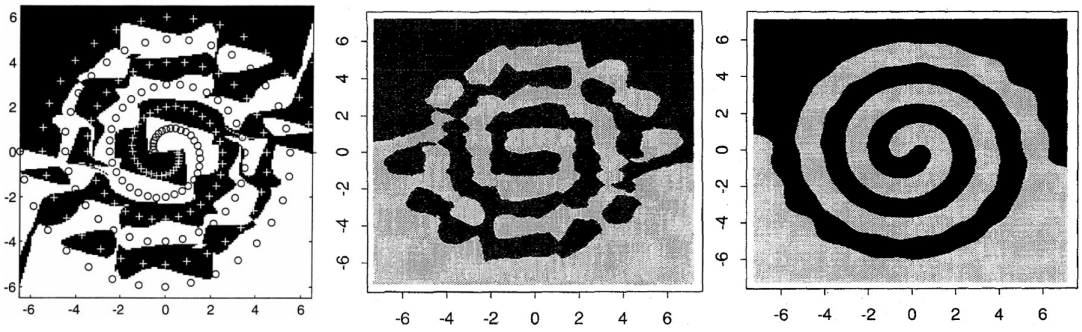
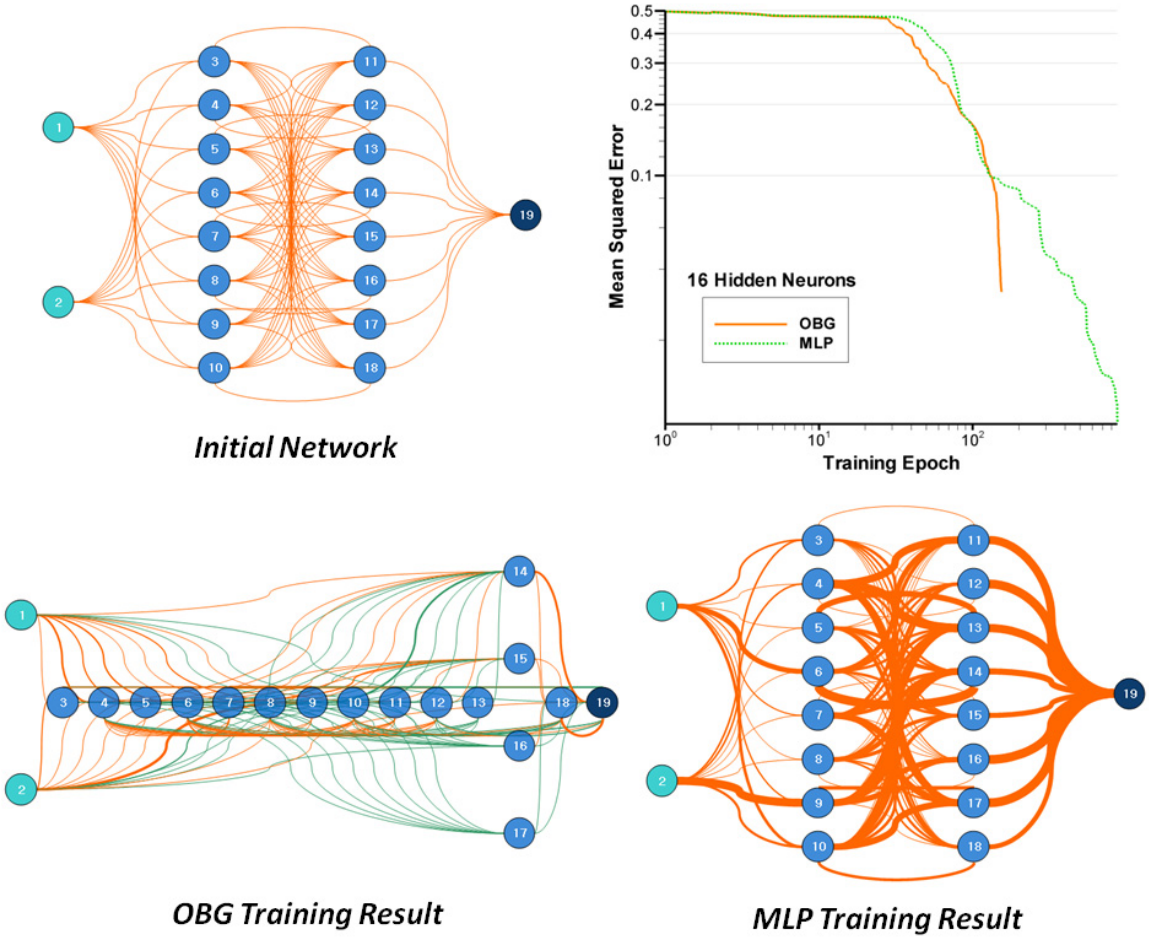


Figure 46: Solutions of Two-Spiral Problem by Cascade Correlation (left) and Projection Pursuit learning networks (middle, right)[1],[44]



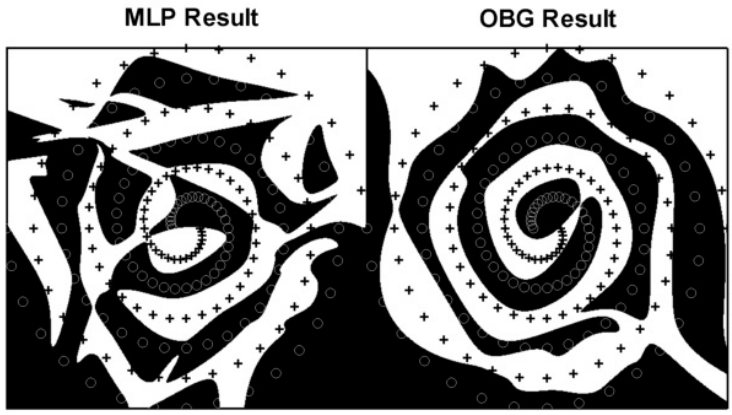


Figure 47: One example from the results of Experiment 2

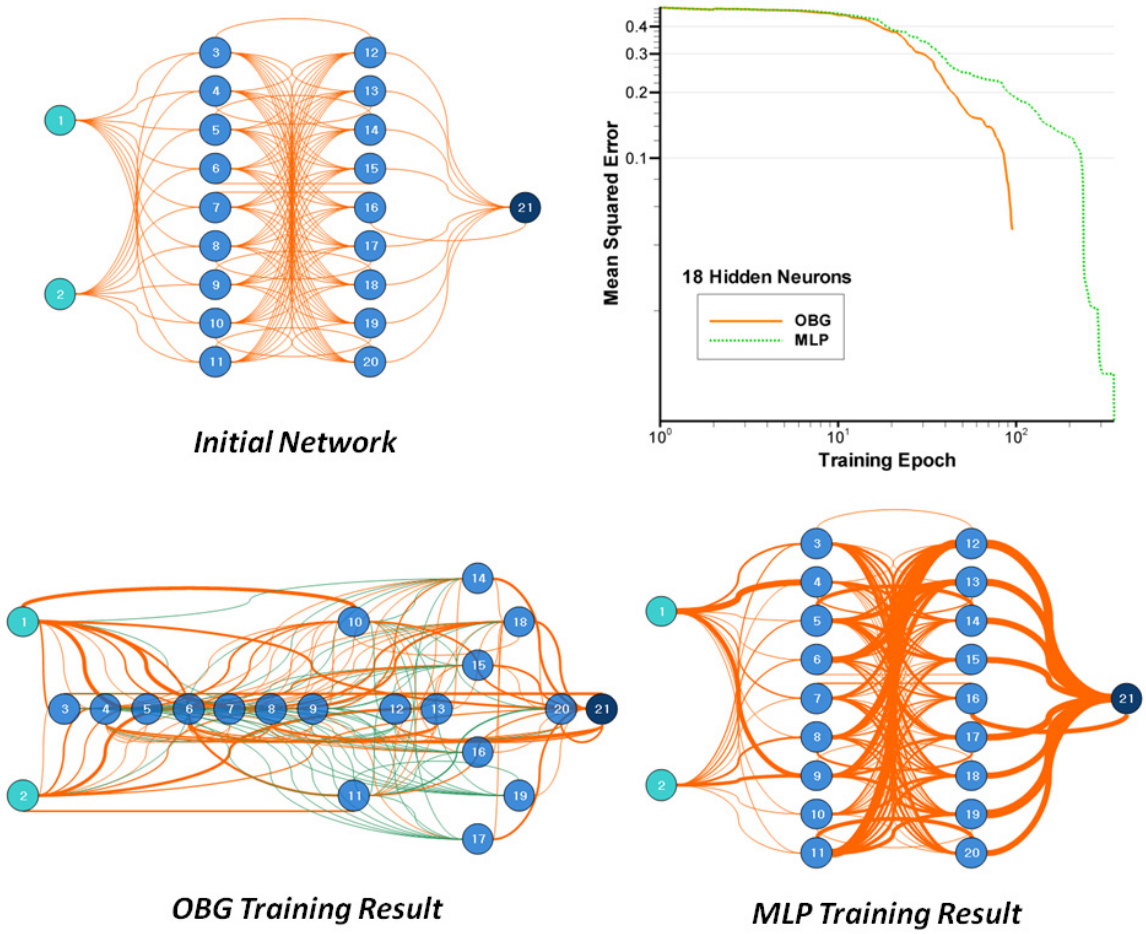
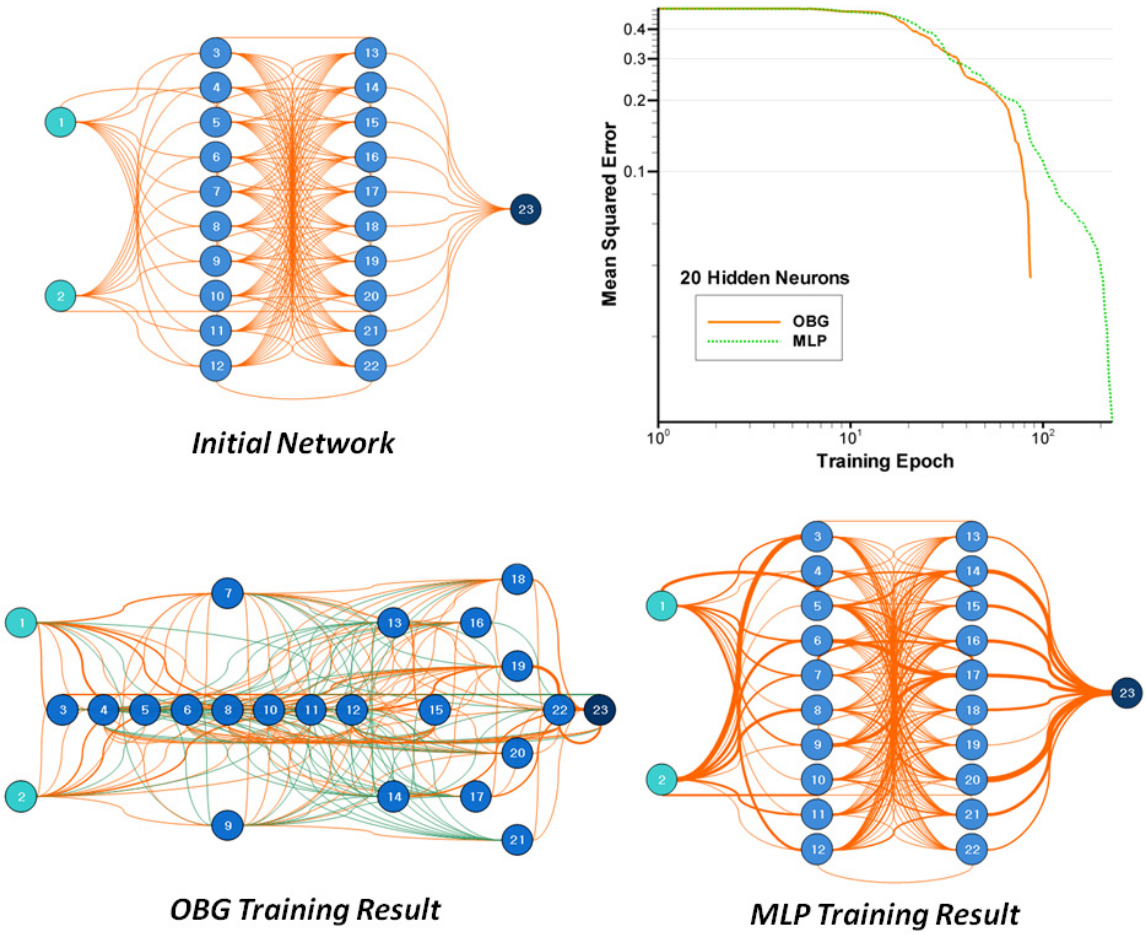


Figure 48: Another example from the results of Experiment 2



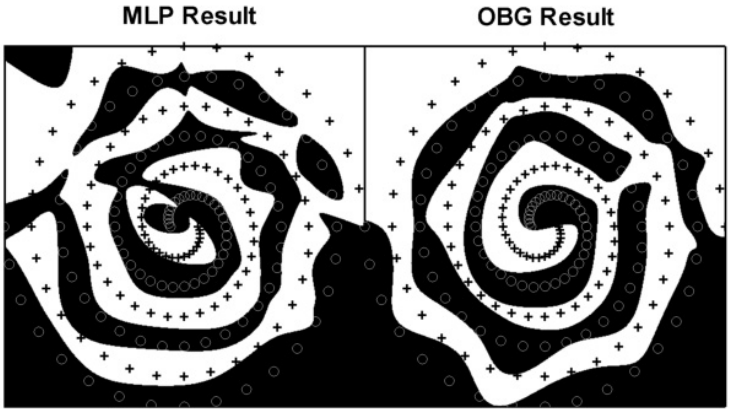


Figure 49: Another example from the results of Experiment 2

5.3 Third Experiment; Generalization Performance

Previous two experiments have been executed to probe the basic functionality of the OBG algorithm as a integrated mechanism to simultaneously learn better weights and connectivity to enhance training performance. In these experiments, the measure for the training performance was simply amount of error reduction and the corresponding number of training epochs. For a simple regression problem to surrogate an elementary fluid mechanics analysis of cylindrical pipe flow and a classical classification task of the Two-Spiral Problem, the OBG algorithm outperformed the conventional MLP networks within the common BP-LM optimization scheme.

As the third experiment, the generalization performance of the OBG will be compared to the MLP network's using separated data sets for validation and testing. In the conventional training method for the MLP networks, the given training examples are, usually, randomly divided into three groups to ensure the generalization performance of the training result;

- Training set is shown to the network which will adjust its parameters to reduce discrepancy between its outputs and target values.
- Validation set is used to calculate *validation error* which is monitored throughout the training procedure and affects the termination criteria of the training procedure.
- Testing set is used to calculate *testing error* after the training ends as a measure of the generalization performance of the network.

For the same regression problem as the first experiment, a series of trainings has been performed to compare the validation and testing errors of the conventional BP-LM and the OBG training with following setup;

- The initial network has 4-6-6-1 configuration with randomly initialized weights. The exactly identical initial networks are used to both MLP and OBG trainings.

- Although the successive deterioration in the validation error enforces the termination of the training in the conventional MLP training, the termination action has not been enforced in this experiment to observation purpose until the number of training epochs reach $10,000$.

- Considering the uncertainty depending on both the random initial parameters and the random selection of training, validation, and testing data, repetition of 10 trainings are executed for each case.

- At each training campaign, the final training result is defined at the training epoch where the validation error is in its minimum value.

- Out of the total 522 training examples, 70% of the data is used for network training, 15% to calculate validation error, and 15% to calculate independent testing error.

Table 3 summarizes the results from 10 trainings. The training result for each case has been extracted when the validation error is at the minimum during the limited $10,000$ training epochs. Therefore, the training error and testing error are also affected by the validation error in each case. Overall result shows that either one training method of MLP or OBG has no distinctive advantage in the generalization performance.

Figure 50-57 are the initial and final network configuration with error history for each case. These individual data provides following observations;

- Both for the MLP and the OBG trainings, there is a strong correlation between the validation error and testing error showing that both of them are useful indicator of the generalization performance of the network.

Table 3: Result of Experiment 3

Cases	Epochs*		Training**		Validation**		Testing**	
	MLP	OBG	MLP	OBG	MLP	OBG	MLP	OBG
1	282	3050	5.07	**5.99**	3.64	**4.97**	3.43	**4.39**
2	152	1390	4.39	**5.44**	3.48	**3.62**	3.20	**3.54**
3	1147	3243	**6.28**	6.19	**6.02**	4.57	**5.95**	4.22
4	41	3202	4.30	**5.95**	3.34	**4.30**	3.08	**3.90**
5	1025	23	**5.44**	4.12	**3.66**	3.37	**3.54**	3.09
6	311	1075	5.15	**5.83**	**4.21**	3.85	**3.78**	3.39
7	1217	3924	5.53	**6.03**	3.82	**4.60**	3.44	**4.32**
8	69	5992	5.36	**5.71**	**5.19**	4.09	**4.67**	3.54
9	5795	768	**5.46**	5.45	3.61	**4.72**	3.49	**4.63**
10	1216	69	**5.05**	4.64	**3.78**	3.77	**3.65**	3.43

*Number of epochs to the minimum validation error.
**Errors in $-\log_{10}(MSE)$; the bigger, the less error.

- The trajectory of validation error contains many local minima and the proper detection of the minimum validation error might have to be more sophisticated than the conventional way of such as counting the number of successive deterioration in this error value.

- Despite the increased number of the connections in the network than in the MLP training, the OBG training does not necessarily result in the worse generalization performance than the BP-LM training of the MLP networks which has less number of connections.

Therefore, the general result from the current experiment tells us that the application of the OBG training method does not guarantee the improvement in the generalization performance as much as it does in the training-only performance shown in the previous two experiments. The obvious limitation of the current experiment is that only one initial network configuration has been considered. The next experiment extends this limitation by considering more diverse initial network configurations.

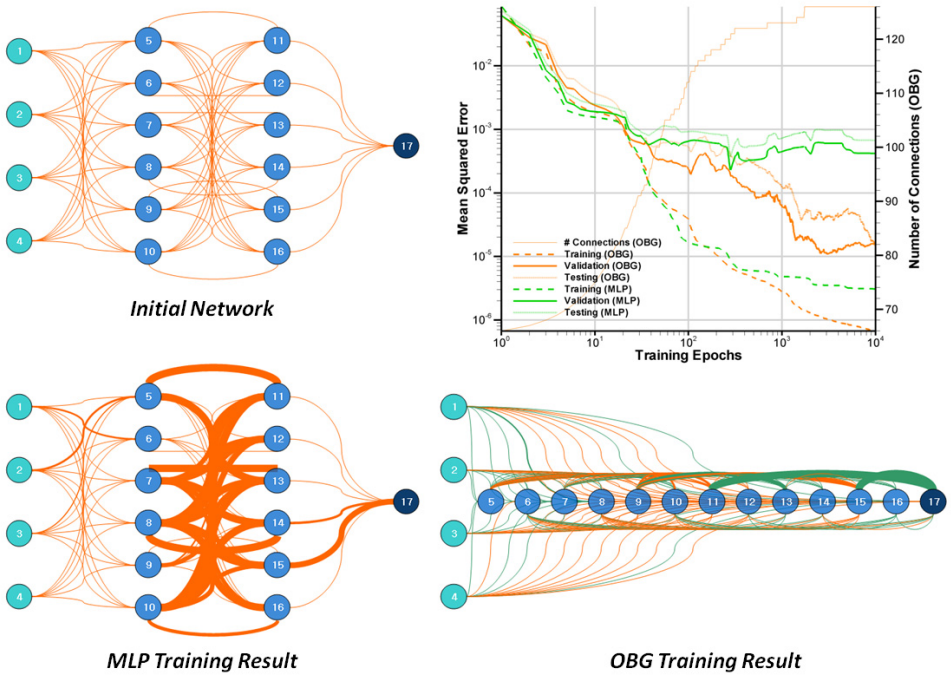


Figure 50: Generalization performance comparison 1, Experiment 3

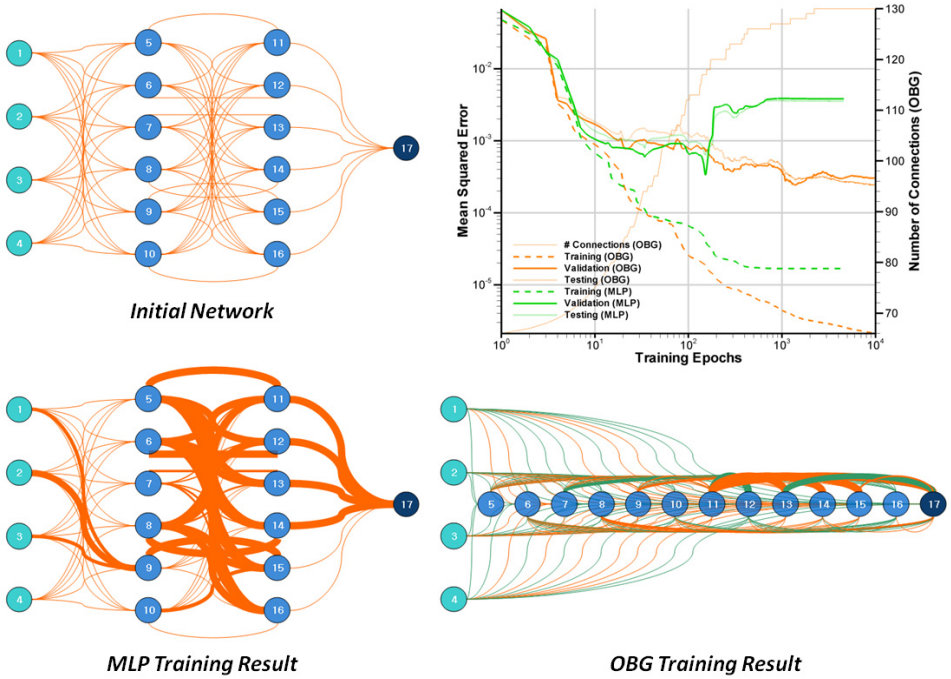


Figure 51: Generalization performance comparison 2, Experiment 3

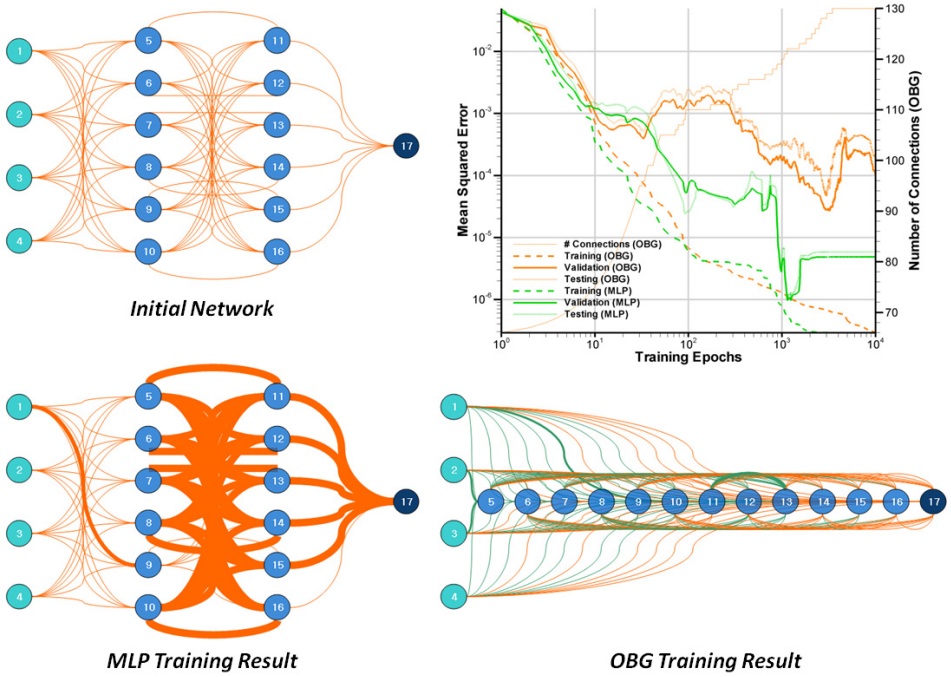


Figure 52: Generalization performance comparison 3, Experiment 3

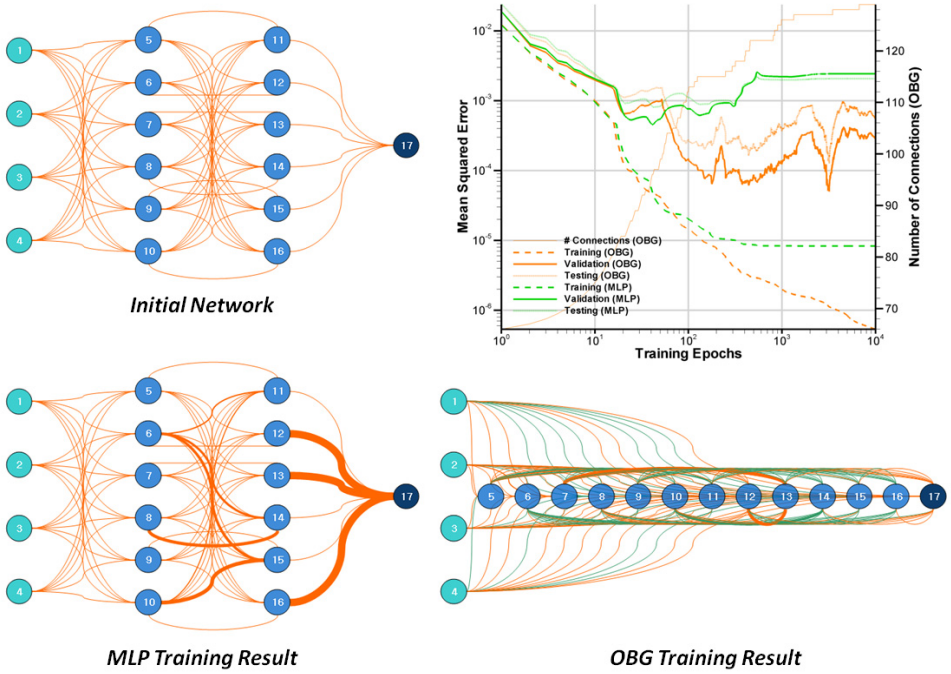


Figure 53: Generalization performance comparison 4, Experiment 3

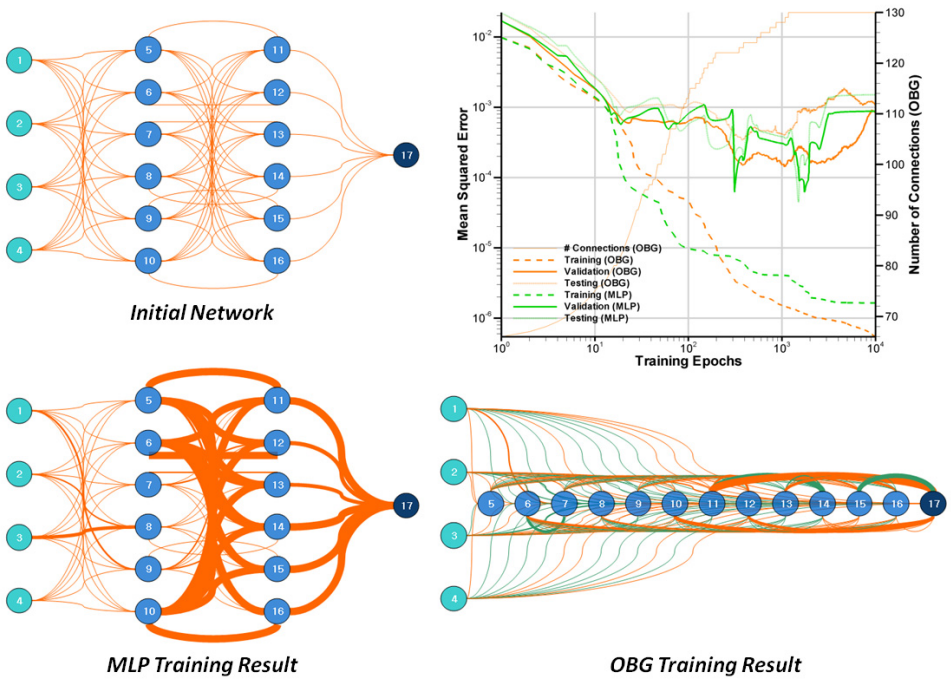


Figure 54: Generalization performance comparison 6, Experiment 3

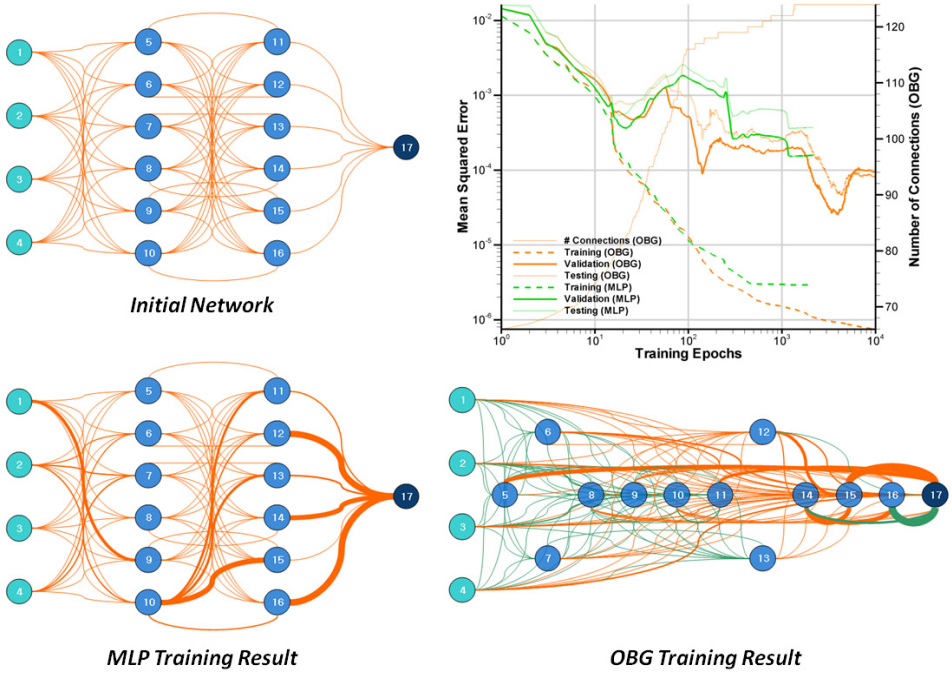


Figure 55: Generalization performance comparison 7, Experiment 3

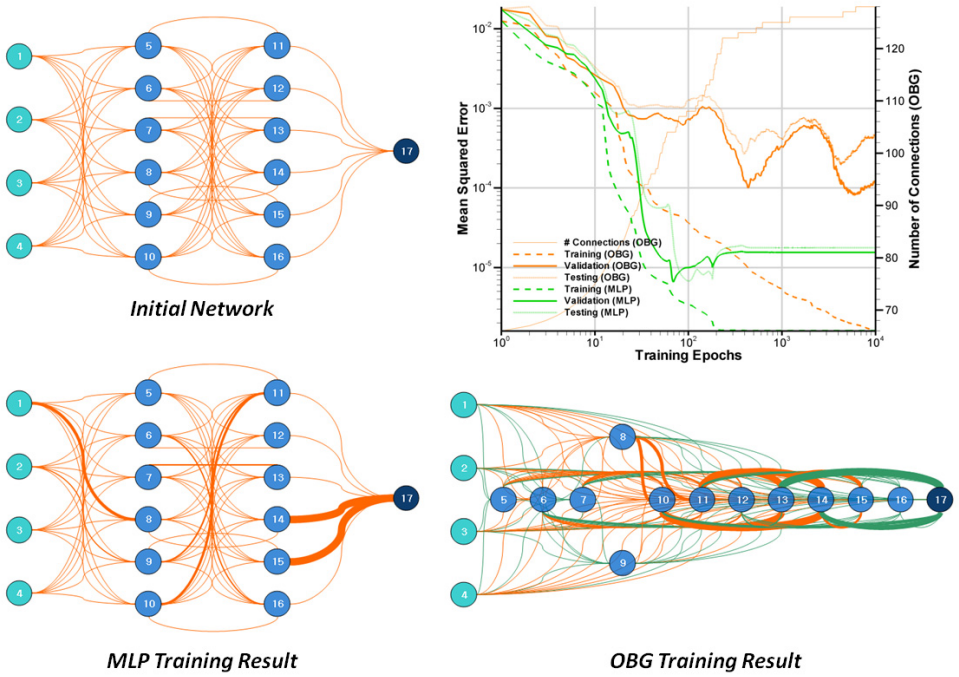


Figure 56: Generalization performance comparison 8, Experiment 3

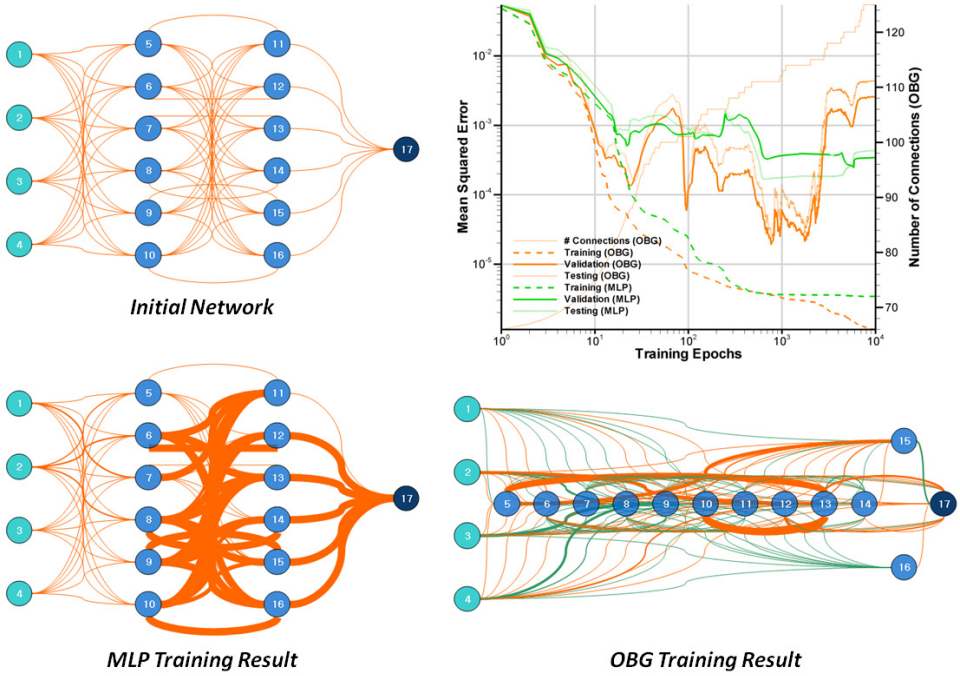


Figure 57: Generalization performance comparison 9, Experiment 3

5.4 Fourth Experiment; Random Networks

As an extension to the previous experiment to observe the generalization performance of the OBG training algorithm, the OBG training performance starting from the randomly connected initial network structures have been compared with the conventional BP-LM training for the MLP networks. One of the ultimate solutions to the general network design problem might be a successful convergence to a useful network solution by starting randomly initialized connection structure as well as randomly initialized weight parameters. Theoretically, the OBG algorithm facilitate towards this goal due to its capability to adjust connectivity as well as weights of the network starting from the minimally connected network. Hence, the current experiment has dual purposes;

- To test generalization performance of the OBG for the variety of initial network configurations and compare them with the conventional MLP networks
- To test the benefit of the randomly connected structures as initial networks compared to the training cases starting from the conventional MLP networks

For this purpose, one simple regression problem to model the Complicated Interaction Regression Function (CIRF, [44]) has been chosen, which has the following definition [44];

$$g(x_1, x_2) = 1.9\left(1.35 + \exp^{x_1}\sin\left(13\left(x_1 - 0.6\right)^2\right)\exp^{-x_2}\sin\left(7x_2\right)\right). \qquad (80)$$

In literature, this problem served as the benchmarking problem for several training algorithms including Cascade Correlation and Projection Pursuit learning networks [44], one of which results has been shown in Figure 59.

In Figure 59, (a) corresponds to the training target consisting of 250 regularly spaced grid points and (b), (c), (d) correspond to the training results for the 10-hidden neuron BP-LM (b), the 10-hidden neuron Cascade Correlation network (c), and the 5-hidden neuron supersmoother-based Projection Pursuit network (d).

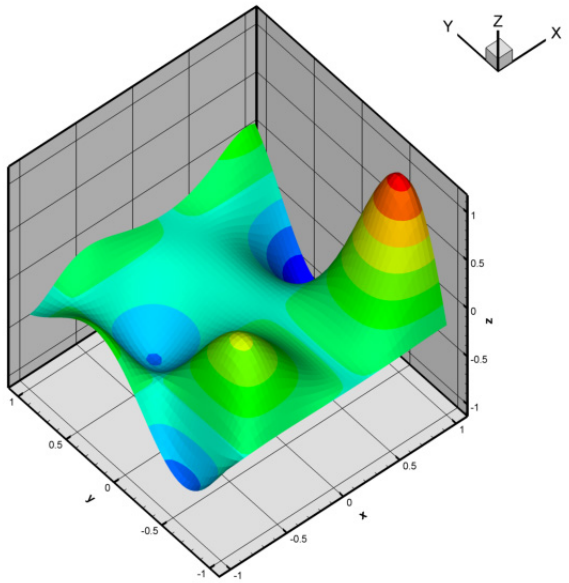


Figure 58: Nonmimensionalized visualization of Complicated Interaction Regression Function

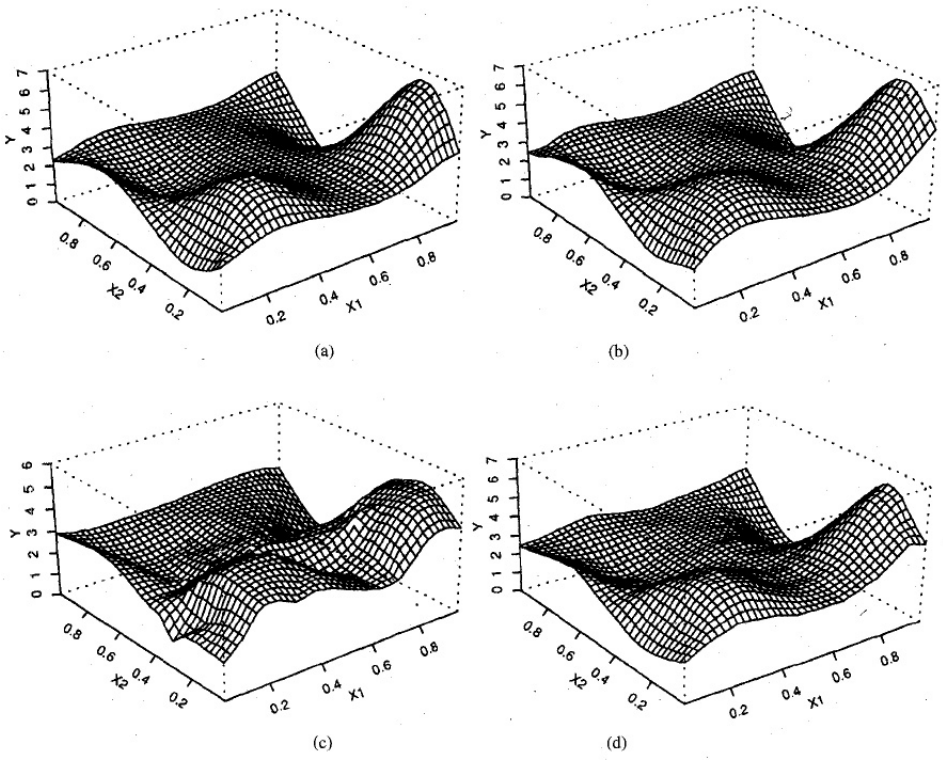


Figure 59: Training target (a) and neural network modeling results (b,c,d)

In the current experiment, 250 regularly spaced points have been used as training data. Among them, 25% of samples have been randomly chosen for validation set and 10% for independent testing error measurement, in each training case. Following matrix of initial network configurations has been used for the comparative study of the MLP and the OBG trainings;

- Single-hidden layer MLPs having 4, 6, 8, 10, 12, 14, 16, 18, and 20 hidden neurons (9 networks)
- Double-hidden layer MLPs having 2, 3, 4, 5, 6, 7, 8, 9, and 10 hidden neurons in each hidden layer (9 networks)
- Randomly connected 4, 6, and 8 hidden neuron networks which have been assigned with the probability to make connection between arbitrary two neurons are 30%, 40%, 50%, and 60% (12 networks)
- Randomly connected 10, 12, 14, 16, 18, and 20 hidden neuron networks with the probability to connection of 20%, 30%, 40%, 50%, and 60% (30 networks)
- Each initial network configuration has been trained 20 times compensating for the uncertainty depending on the random weight initialization and the random selection of training (65%), validation (25%), and testing (10%) examples making total $1,200$ training cases using above 60 initial network configurations.

In the current experiments, the same strategy with the previous experiment has been adopted for the role of validation samples, i.e., the final training result is defined at the epoch whose validation error is in its minimum value. Figure 60 shows the randomly connected initial network and its final training result for three example cases in case of the OBG training. In this diagram, ineffective neuron whose output does not contribute to the output node has been indicated with gray color. These examples show that starting from the sparsely and randomly connected networks have

no problem to be trained towards more densely connected working networks using the OBG algorithm.

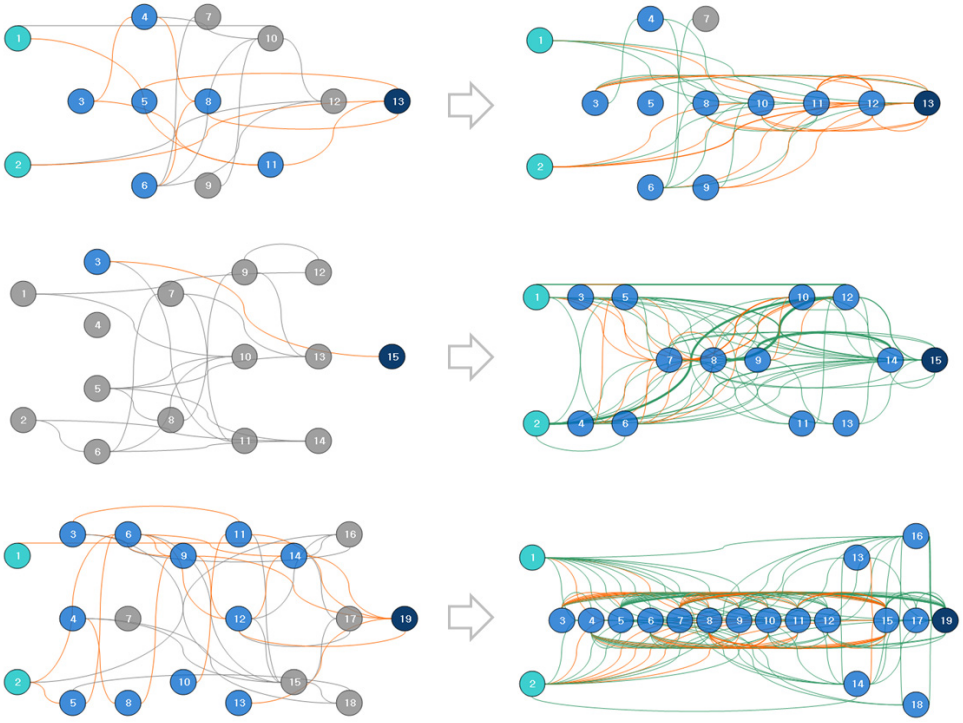


Figure 60: Examples of network configurations, Experiment 4

The training results from all 1,200 individual training cases are plotted in Figure 61. The ordinate in this chart corresponds to the mean squared error in the reverse direction indicating the higher position is for the more accurate training result. The abscissa is the number of optimization parameters which is the sum of the number of initial connections and biases in the network. Analyzing this chart provides following observations;

- For the lower number of optimization parameters (hence, lower connectivity networks), the OBG training starting with randomly connected networks outperform the training cases using MLP networks.
- However, there exists an interval where the double-hidden layer MLPs result in

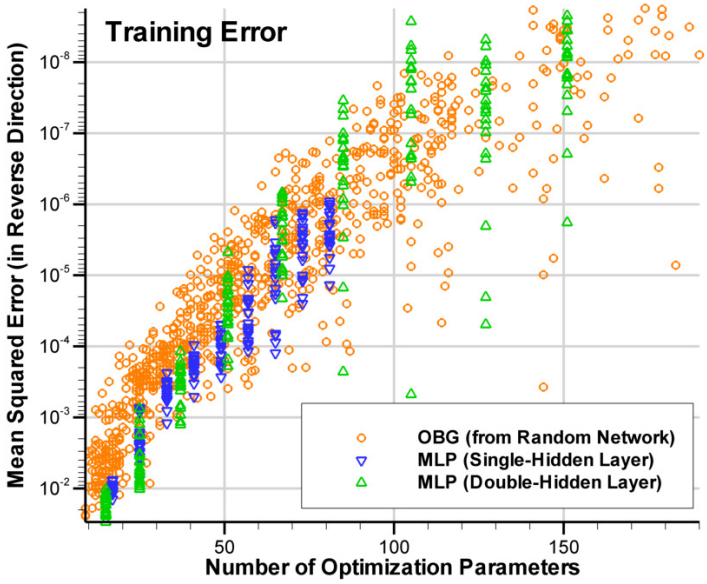


Figure 61: Training error comparison, Experiment 4

the better training accuracy for the given number of optimization parameters.

Similar observation is also true for the validation and the independent testing errors which are shown in Figure 62 and Figure 63.

The current experiment to probe the generalization performance of the OBG trained networks using the variety of initial network configurations of not only conventional MLP networks but also randomly connected networks gives the consistent result with the previous experiment; the improvement in the training and generalization performance by using the OBG algorithm is more evident in cases of small networks especially compared to the single-hidden layer MLP networks. As an effort to extend this improvement to the larger network's case especially when it compared to the double-hidden layer MLP networks which have relatively abundant number of hidden neurons, the next experiment has been performed.

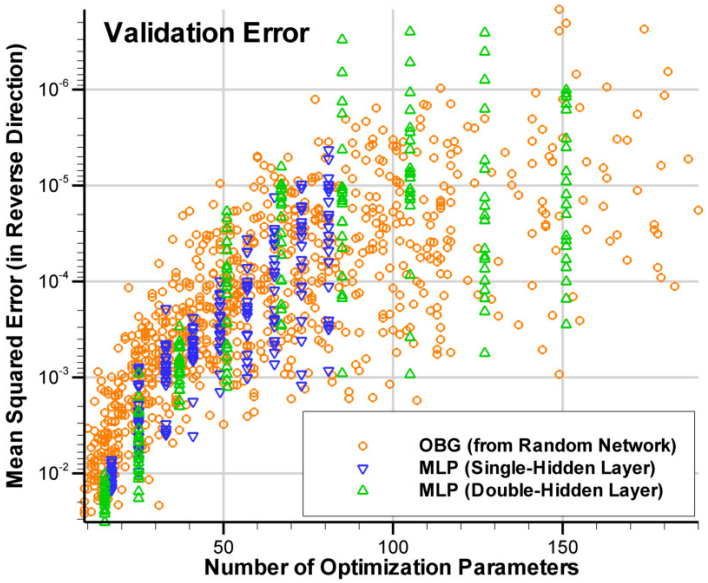


Figure 62: Validation error comparison, Experiment 4

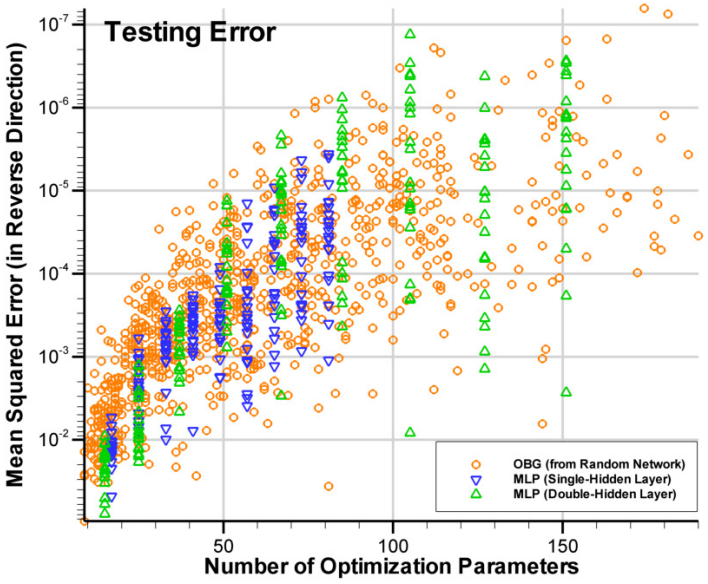


Figure 63: Testing error comparison, Experiment 4

5.5 Fifth Experiment; Additional Neurons and Growth Criteria

Through the previous four experiments, the OBG algorithm proved its potential in the improvement of the training performance compared to the conventional BP-LM algorithm with MLP networks consuming approximately identical optimization time and memory. But its generalization performance shows no significant advantage when it compared to the double-hidden layer MLP networks which have large number of hidden neurons. In the current experiment, first, the OBG algorithm with neuronal decomposition is tested in its network sizing capability. Next, the OBG algorithm has been extended to allow the growth by the correlation-based growth rule resulting in the OBG+ scheme to enhance its generalization performance and, later, a combination of the BP-LM training and the OBG training is tried for the same objective.

5.5.1 Validation Error and Network Sizing

The usefulness of the neuronal decomposition as a mean for local reinitialization has been observed in the objective of finding the optimally-sized networks. The one important heuristics adopted in majority of the neural network applications is to exploit the validation error to determine the point of early termination during the training campaign. This practice is based on the observations that the generalization performance for the previously unseen input data does not monotonically increase as the training performance does during the weight optimization process. By using the separated data for the validation purpose and stopping the training process when the validation error increases, the reasonable generalization performance can be obtained for the given size of the networks [70]. On the other hand, for the given set of the training examples, the generalization performance depends also on the overall size of the networks, i.e., too small networks cannot learn the relationship between the input and the output data while too large networks just memorize the direct mapping for

each training sample resulting in the poor generalization performance with over-fitting problem [51]. Because the conventional paradigm of the neural network training assumes the given initial and permanent network configuration with the fixed size, the search for the optimally-size network requires trial-and-error approach. One appealing idea behind the current experiment is to exploit the OBG training method to find the optimally-sized network and compare its performance and efficiency with the conventional approach. In this experiment the CIRF modeling problem which has been introduced in the previous section is used. The training examples were extracted from the regularly-spaced 10-by-10 grid on the problem domain and the validation examples were located at the center points in the unit cells on that grid system. Figure 65 shows a visualization of the CIRF and the selected samples for the training and the validation purposes. To assess the performance of the OBG training in the aspect of ability to find the optimally-sized networks, first, a *bird-eye view* for the generalization performance versus network size needs to be obtained. Total 18 MLP networks consisting of 9 single-hidden-layer MLPs (having 4, 6, 8, 10, 12, 14, 16, 18, and 20 hidden units) and 9 double-hidden-layer MLPs (having 2, 3, 4, 5, 6, 7, 8, 9, and 10 hidden units in each layer) have been used to this purpose. The repeated 5 trials to the minimum validation error for each network configuration provided the required information on relation between the minimum validation error and the network sizes as depicted in Figure 64.

The general trend is clearly identifiable and there exists the region of the optimally-sized networks in terms of the minimum validation error. Beyond the magnitude of the validation error, the generalization performance or quality can be visualized by the difference between the target values and the network predictions such as shown in Figure 66 and 67. In these figures, while the modeled output surfaces are less differentiable, the prediction capability for the previously unseen region is greatly

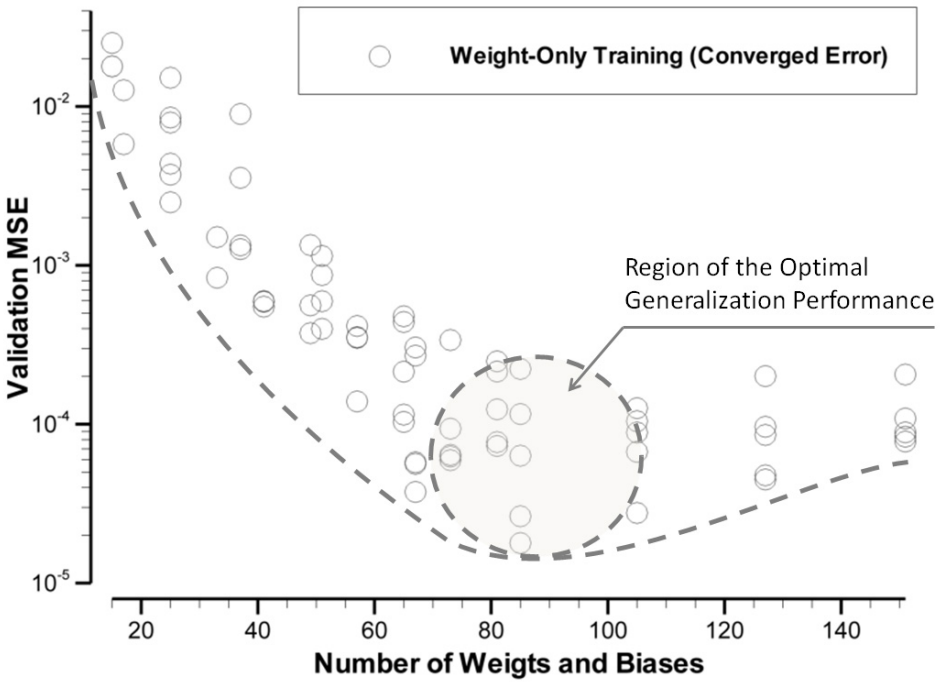


Figure 64: An example of training error variation during the weight-only training and the OBG training (4 additional neurons allowed)

dissimilar. The severe distortion of the error surface in Figure 66 represents a significant over-fitting problem.

Now, the OBG training can be executed from small initial networks and its convergence trajectory can be mapped onto Figure 68. Three such trajectories has been drawn in the figure. These are ones of the typical OBG training results started from 2-5-1, 2-4-4-1, and 2-5-5-1 MLPs and converge to the region of the optimally-sized networks. Hence, the OBG has the potential to become a useful tool in the network sizing problem for the maximum generalization performance in case of being numerically competitive to the existing methods. Table 5.5.1 shows the computation time for all training cases. Even though the single run time for MLP training cases is much lower than the computation time required for the OBG training to converge to the minimum validation error, the total required time for a network designer to

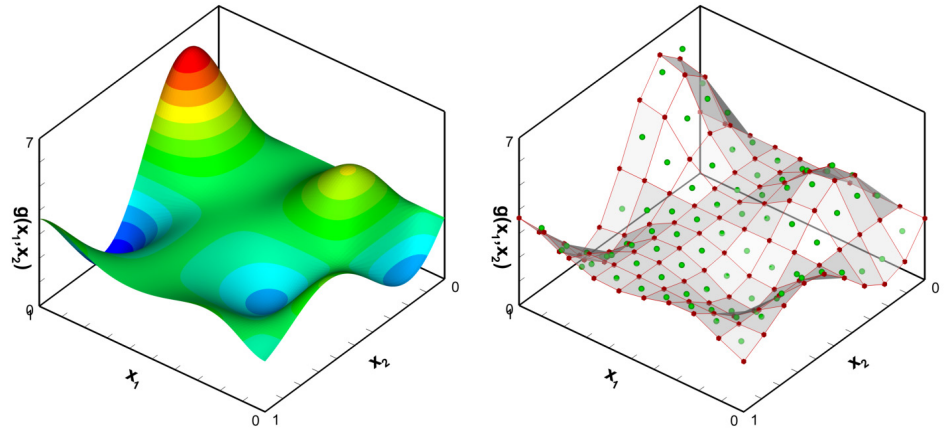


Figure 65: Visualization of the Complicated Interaction Regression Function (left), 10-by-10 training grid and 9-by-9 validation grid (right)

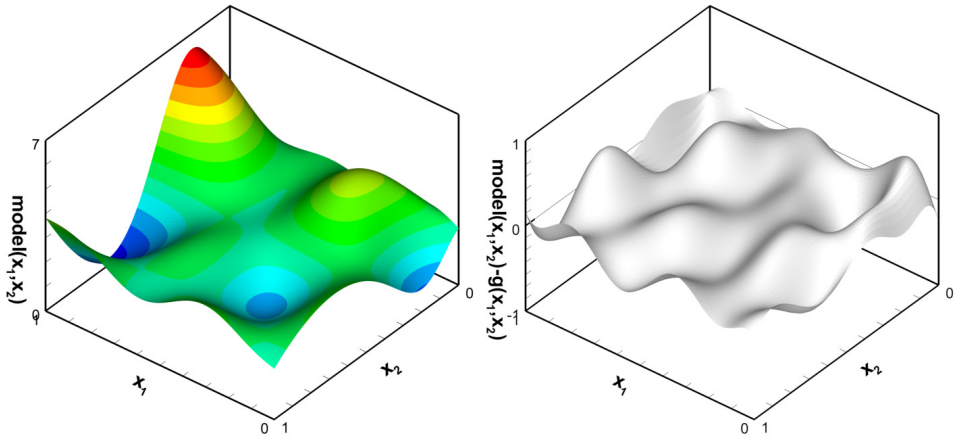


Figure 66: One example of the poor generalization performance, model surface (left) and the difference from the true surface (right)

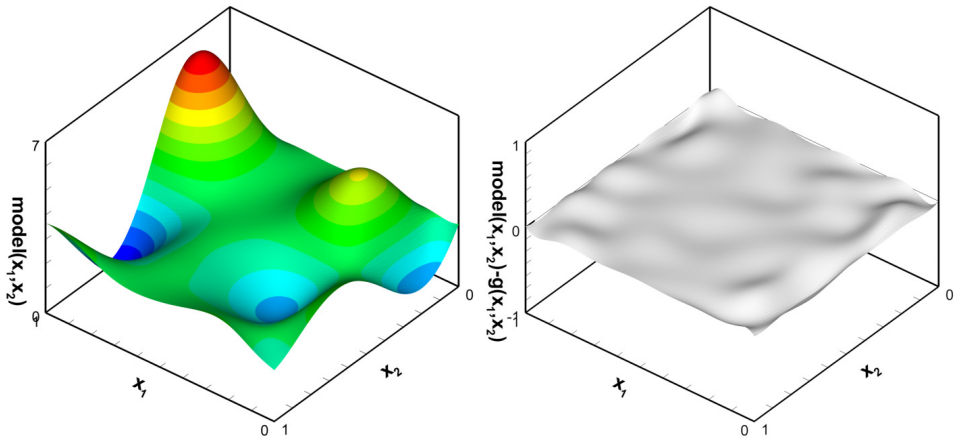


Figure 67: One example of the good generalization performance, model surface (left) and the difference from the true surface (right)

appropriately size the network in the objective of the maximum generalization performance must be comparable to the total sum of the individual MLP training times. On average, if the run time for an individual MLP training is equal to 1, the total run time accumulated for MLP runs in the current experiment corresponds to approximately 100. Applying this scale to the three OBG runs, the run time for single OBG training has the order of 10. In other words, to obtain such a macroscopic information depicted in Figure 68 which is critical to the proper sizing of the networks, the conventional MLP training method requires 100 times more computation than the single training case but the OBG training needs only 10 times more computation. Of course, the wise strategy, instead of brute-force trial-and-error approach, to reach to the optimal sizes of networks can reduce the computation time in both training methods. The three cases of the OBG training results show that the computation time is significantly reduced as the initial network size approaches closer to the optimal size.

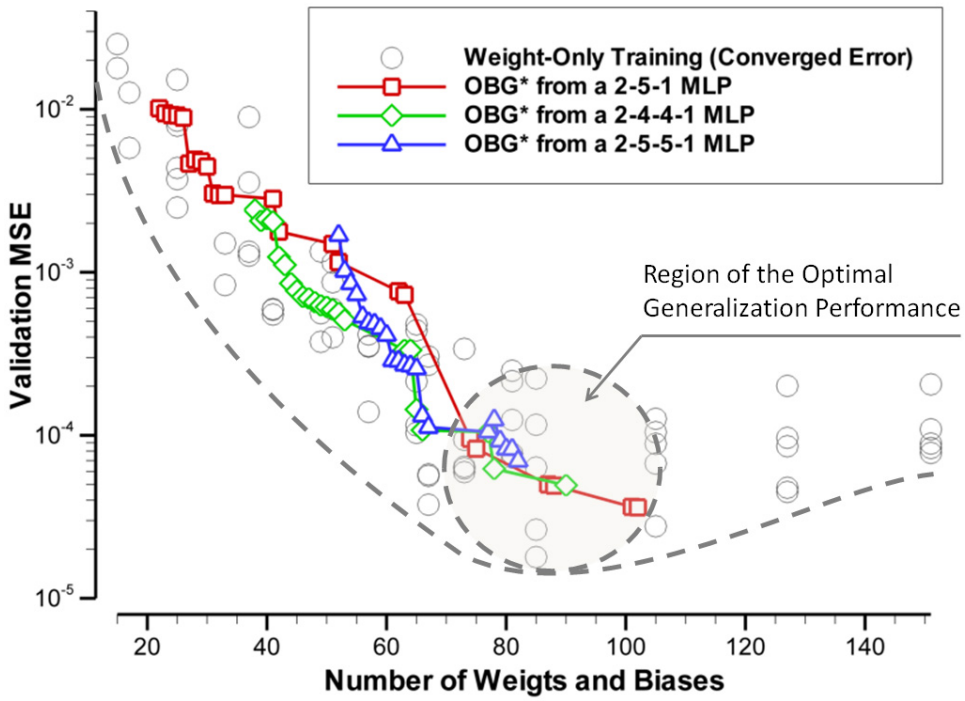


Figure 68: Validation error space and taining results from the two training methods

Table 4: Summary of computation time for CIRF Problem (in Seconds)

Hidden	4	6	8	10	12	14	16	18	20
MLP(1L)*	0.1	0.2	0.2	0.4	0.3	0.6	2.5	3.9	3.0
MLP(2L)*	0.4	0.4	0.7	1.1	4.0	4.9	5.3	10.3	9.3
MLP Total	240								
OBG1	35 (from 2-5-1 MLP)								
OBG2	15 (from 2-4-4-1 MLP)								
OBG3	4.0 (from 2-5-5-1 MLP)								

*Average computation time of 5 repeated trials.

5.5.2 OBG+ Scheme for Adding Neurons

Traditionally, the addition of computational units has been executed when the final training result is unsatisfactory such as in the dynamic node creation [90], Cascade Correlation [24], and Projection Pursuit [44]. In these methods, depending on the learning algorithm, the further training proceeds with the additional neuron or entire training procedure is restarted. While the significant reduction in training time is expected by not repeating entire training procedure, a disadvantage of further training with an additional neuron is that pre-existing network components have already been optimized leaving only little leverage of improvement in the training performance by additional network components. To overcome this shortage, Cascade Correlation builds full connections from all existing neurons to the newly added neuron and Projection Pursuit optimizes activation function of the newly added neuron which is also connected to all nodes in the adjacent layers. In case of the OBG training method, more natural criterion is applicable to decide when to add a new computational unit without the need for re-training due to its flexibility to adjust connectivity of network as a part of weight optimization procedure.

5.5.2.1 Concept of Connectivity Ratio

Every feed-forward network has its maximum number for the allowable connections as defined in the previous chapter. The *connectivity ratio*, r_{conn}, of a feed-forward

network is defined as

$$r_{conn} = \frac{C_{conn}}{C_{forward}} \tag{81}$$

where C_{conn} is the number of current connections (regardless of whether the corresponding connection is adjustable or frozen) and $C_{forward}$ is the allowable maximum number of feed-forward connections determined by the network configuration defined by such as number of input, output, and hidden nodes. During the OBG training procedure, the connectivity ratio is monotonically increased as the network grows towards more connectivity. One possible criterion to execute addition of new computational unit is to set the maximum limit on the connectivity ratio and to add a new neuron whenever the connectivity ratio reaches or exceeds this limit. Using the pre-described algorithm on the additional neuron, the new neuron will be randomly positioned in the network between the pre-existing neuron whose output has the maximum correlation to the error of output node whose error is the largest among other output nodes and that output node. And only two additional connections will be established linking the new neuron to the pre-existing network components. As soon as the new neuron is added the connectivity ratio is decreased responding to the increased number of possible feed-forward connections by the additional neuron in the network. This particular scheme is called the OBG+ scheme.

5.5.2.2 Comparison of MLP, OBG, and OBG+ Trainings

Using the problem to fit *Complicated Interaction Regression Function* introduced in the previous section, the comparative trainings have been performed for the BP-LM training on MLP networks, the OBG, and the OBG+ training method. Following initial network configurations have been used;

- 6 single-hidden layer MLPs with 10, 12, 14, 16, 18, 20 hidden neurons
- 6 double-hidden layer MLPs having 5, 6, 7, 8, 9, 10 hidden neurons in each hidden layer

To capture generic trend in the training results affected by the random initial parameters and the random selections on the training, validation, and testing samples, each configuration has been trained 10 times, resulting in total 120 training cases for each training method. In this experiment, 15% of training examples has been used for validation and 10% for the measurement of independent testing error. Again, the final training result is obtained at the epoch with the minimum validation error and the further training results are discarded. In the current experiment, the maximum connectivity ratio allowable has been set to the value of 0.75.

Table 5 shows the network growth status after training finished, where each cell contains the averaged value for 10 repeated trainings. The number of neurons and the number of optimizable parameters for MLP networks and OBG trained networks are same but the OBG+ training method results in the increased number of neurons and optimizable parameters. Total number of connections in networks increases during training both for the OBG and the OBG+ training.

Table 5: Network size after training, Experiment 5

Initial Net	N_{neuron}			C_{conn}			$C_{opt} + N_{bias}$		
	MLP	OBG	OBG+	MLP	OBG	OBG+	MLP	OBG	OBG+
2-10-1	13	13	20.5	30	74.00	142.3	41	41	63.50
2-12-1	15	15	19.4	36	98.50	128.8	49	49	62.13
2-14-1	17	17	21.0	42	125.6	151.5	57	57	69.00
2-16-1	19	19	22.0	48	153.0	166.4	65	65	74.00
2-18-1	21	21	22.5	54	165.9	177.3	73	73	77.50
2-20-1	23	23	24.0	60	194.8	199.9	81	81	84.00
2-5-5-1	13	13	20.5	40	76.63	144.8	51	51	73.50
2-6-6-1	15	15	20.4	54	102.1	142.0	67	67	83.13
2-7-7-1	17	17	22.3	70	130.3	169.9	85	85	100.8
2-8-8-1	19	19	24.1	88	163.3	200.4	105	105	120.4
2-9-9-1	21	21	26.1	108	200.6	237.6	127	127	142.4
2-10-10-1	23	23	27.8	130	235.4	271.3	151	151	165.3

Figure 69, 70, and 71 are three examples showing the variation of the network structure and connectivity for three training methods. The required computation time for each training has been measured and Table 6 shows the number of training

epochs, required computation time, and the approximated computation time per each epoch, where each cell value is the average of 10 trials. Figure 72 is the graphical representation of the same data for the comparison of the computation time per training epoch. As expected by the algorithmic properties, the increment of computation time of the OBG training is minimal despite the growth in the total number of connections comparing the OBG method and the MLP network training.

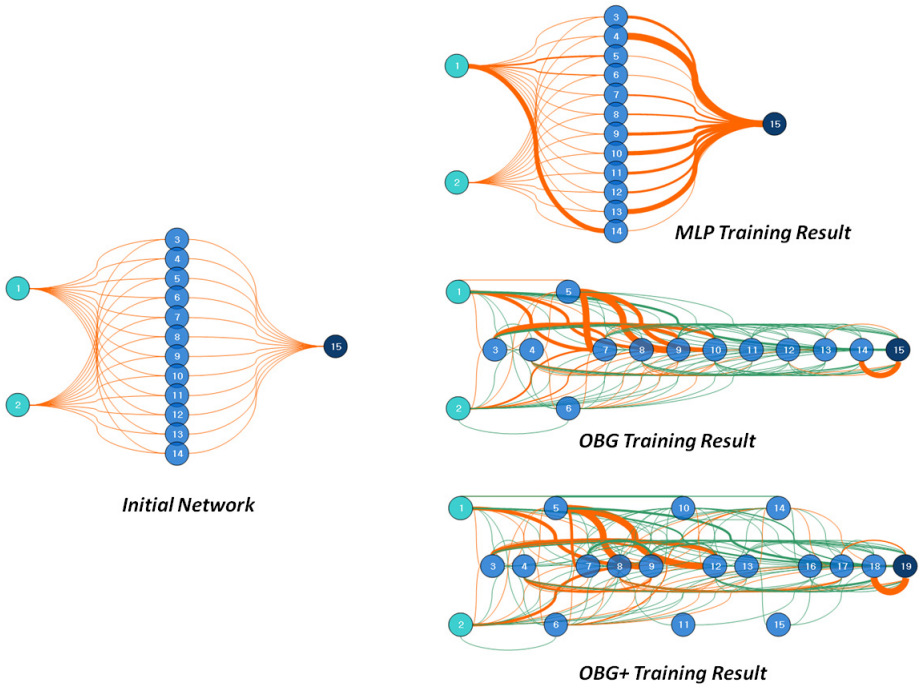


Figure 69: Examples of network configurations, Experiment 5

Figure 73-84 provide the modeling capacities of final networks resulted from the BP-LM method on MLP networks, the OBG training, and the OBG+ training method. Each figure contains two sets of three-dimensional plots for network output values responding to regularly spaced grid consisting of $2,500$ point inputs. The upper row contains plots for the network output itself and the lower row for the difference between the network outputs and the target values. Because only 250 points have been used for the network training procedure, these two types of plots can be used to inspect the quality of modeling, i.e., the generalization performance

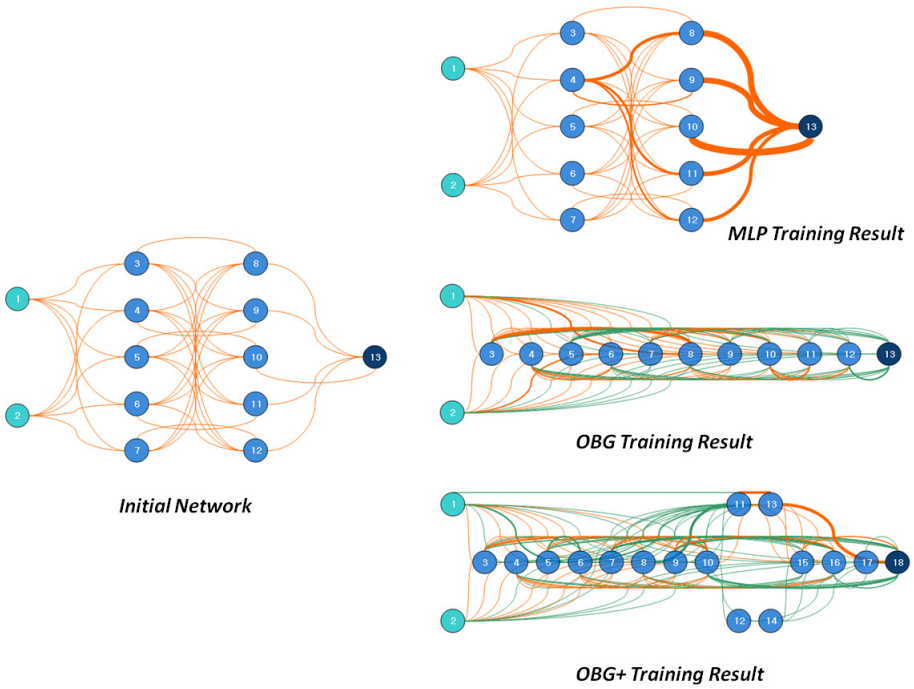


Figure 70: Examples of network configurations, Experiment 5

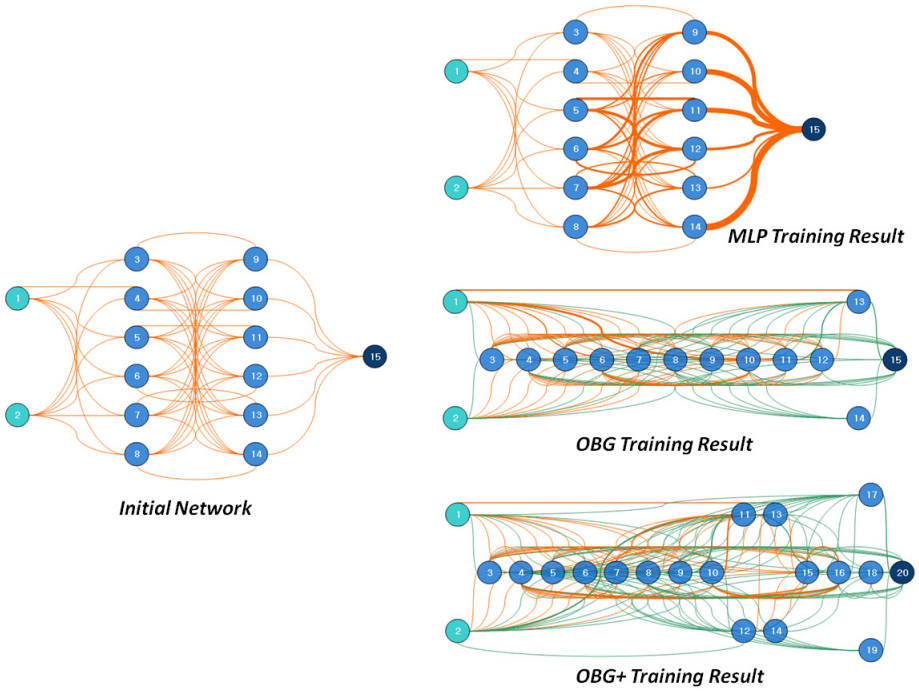


Figure 71: Examples of network configurations, Experiment 5

Table 6: Training time to minimum validation error, Experiment 5

Initial	Epoch*			Second*			Second/Epoch		
Net	MLP	OBG	OBG+	MLP	OBG	OBG+	MLP	OBG	OBG+
2-10-1	6498	9832	13900	13.0	24.8	68.7	0.0020	0.0025	0.0049
2-12-1	1280	10433	12191	3.42	34.7	54.6	0.0027	0.0033	0.0045
2-14-1	7969	12570	14035	29.2	57.1	87.6	0.0037	0.0045	0.0062
2-16-1	7012	12247	7860	30.9	66.8	54.4	0.0044	0.0055	0.0069
2-18-1	9302	15389	18162	54.8	110	141	0.0059	0.0071	0.0077
2-20-1	4032	16310	14308	28.3	138	126	0.0070	0.0085	0.0088
2-5-5-1	5077	6911	13261	13.4	22.0	78.4	0.0026	0.0032	0.0059
2-6-6-1	4039	7037	6599	17.0	34.4	53.4	0.0042	0.0049	0.0081
2-7-7-1	14827	12665	11319	107	102	127	0.0072	0.0081	0.0112
2-8-8-1	13292	12252	13505	153	153	220	0.0115	0.0125	0.0163
2-9-9-1	16937	17526	17176	302	334	424	0.0178	0.0190	0.0247
2-10-10-1	12437	9408	12959	341	271	469	0.0274	0.0288	0.0362

*Number of seconds and epochs to the minimum validation error.

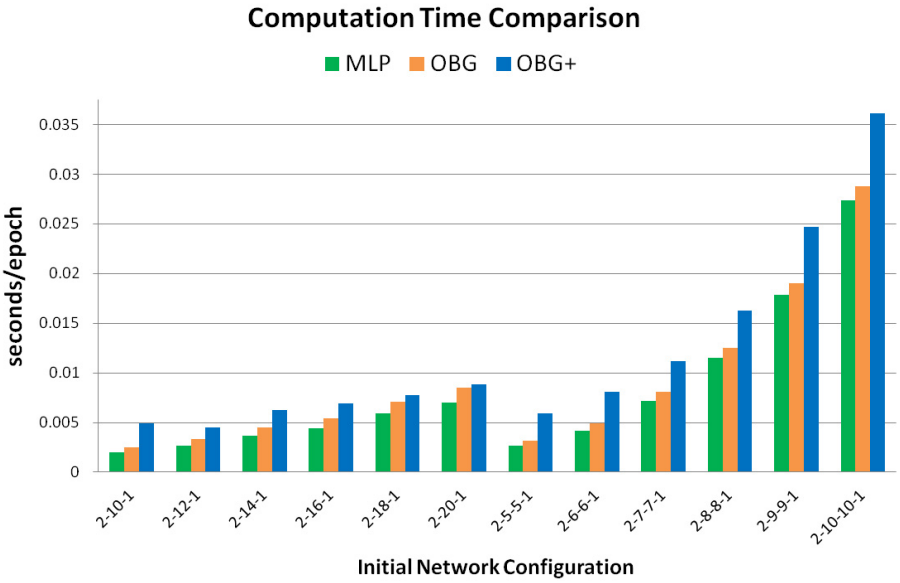


Figure 72: Computation time comparison, MLP, OBG and OBG+, Experiment 5

of resultant networks. Each plot has been constructed by synthesizing 10 network outputs from the repeated 10 training runs with averaged value of them to capture the overall trend in the generalization performance of each training case. If the model is perfect, the *difference* plot has to be flat everywhere but it usually has *wiggles* and *distortions*, especially, near the input boundary areas. As a result of observation on these plots, all the network trained from three different training methods give reasonably accurate fit to the target function but the exaggerated *difference* plots show that there exist evident differences on the detailed training results among the three training methods. For the smaller networks, the generalization performance is enhanced as the OBG and the OBG+ methods are applied compared to the conventional BP-LM training method on the MLP networks. But this enhancement is less evident for the larger networks. Table 7 summarizes overall training results by showing training, validation, and testing error, all of which are the averaged value for 10 training repetitions. Figure 85, 86, and 87 provide graphical representation of the same data given in the table.

Table 7: Training error comparison for MLP, OBG, and OBG+, Experiment 5

Initial Net	Training*			Validation*			Testing*		
	MLP	OBG	OBG+	MLP	OBG	OBG+	MLP	OBG	OBG+
2-10-1	3.77	4.95	**6.55**	3.41	4.42	**5.09**	3.29	3.66	**4.25**
2-12-1	4.15	5.91	**6.30**	3.78	4.77	**5.03**	3.85	3.89	**4.72**
2-14-1	4.47	**6.61**	6.01	3.87	**4.84**	4.60	3.87	4.29	**4.30**
2-16-1	5.28	**6.83**	6.68	4.64	4.95	**5.03**	4.57	4.94	**5.04**
2-18-1	5.80	6.99	**7.72**	**4.65**	4.33	4.42	4.36	5.05	**5.61**
2-20-1	5.91	**7.54**	7.52	**4.72**	4.66	4.60	4.44	**5.52**	5.51
2-5-5-1	4.39	5.83	**7.24**	3.71	4.43	**5.05**	3.75	4.61	**5.20**
2-6-6-1	5.94	**6.77**	6.62	4.91	4.93	**5.29**	4.93	5.23	**5.39**
2-7-7-1	7.57	8.07	**8.55**	5.98	5.75	**5.77**	4.37	4.33	**4.84**
2-8-8-1	8.77	9.28	**9.70**	**6.62**	6.11	6.08	5.87	**5.93**	5.87
2-9-9-1	10.1	**10.5**	10.4	**7.01**	5.79	5.77	5.88	**6.16**	5.41
2-10-10-1	7.00	9.04	**10.6**	**6.19**	5.19	4.96	5.37	**5.76**	5.48

*Errors in $-\log_{10}(MSE)$; the bigger, the less error.

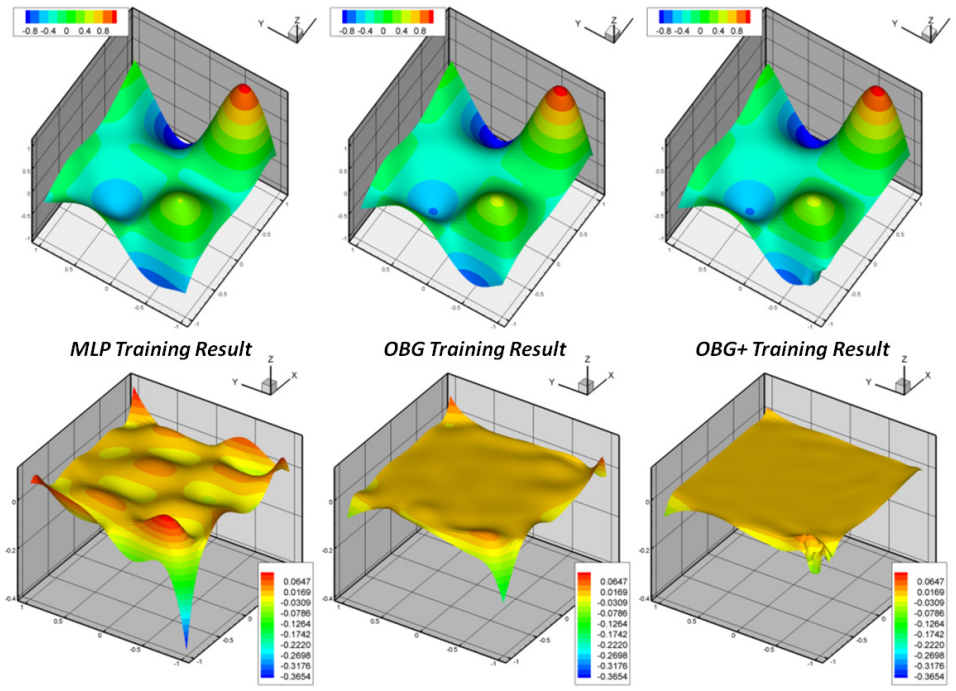


Figure 73: Averaged output of 10 trainings from 2-10-1 network, Experiment 5

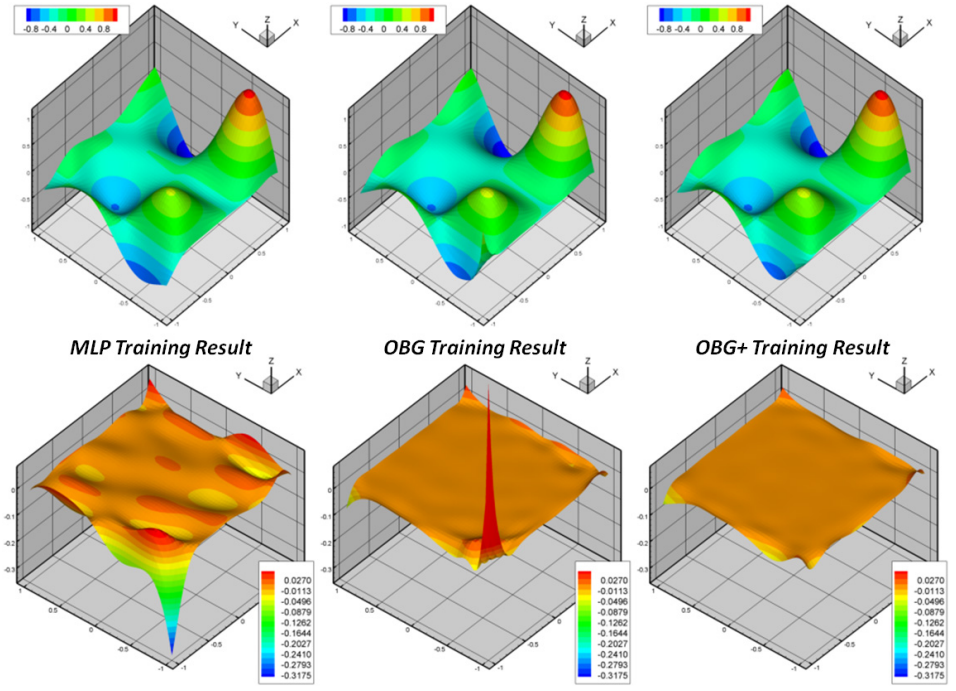


Figure 74: Averaged output of 10 trainings from 2-12-1 network, Experiment 5

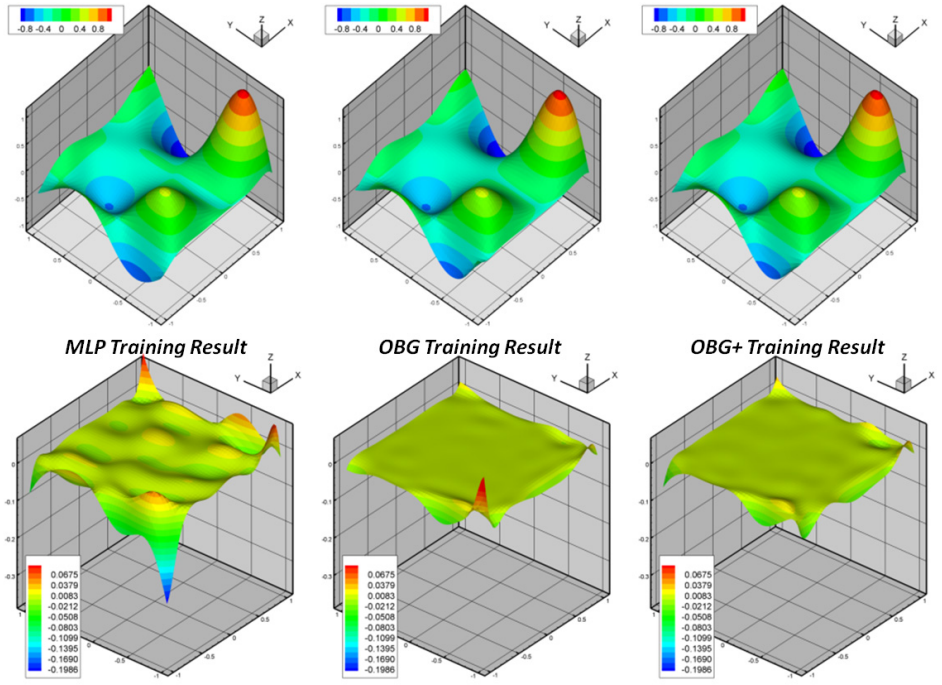


Figure 75: Averaged output of 10 trainings from 2-14-1 network, Experiment 5

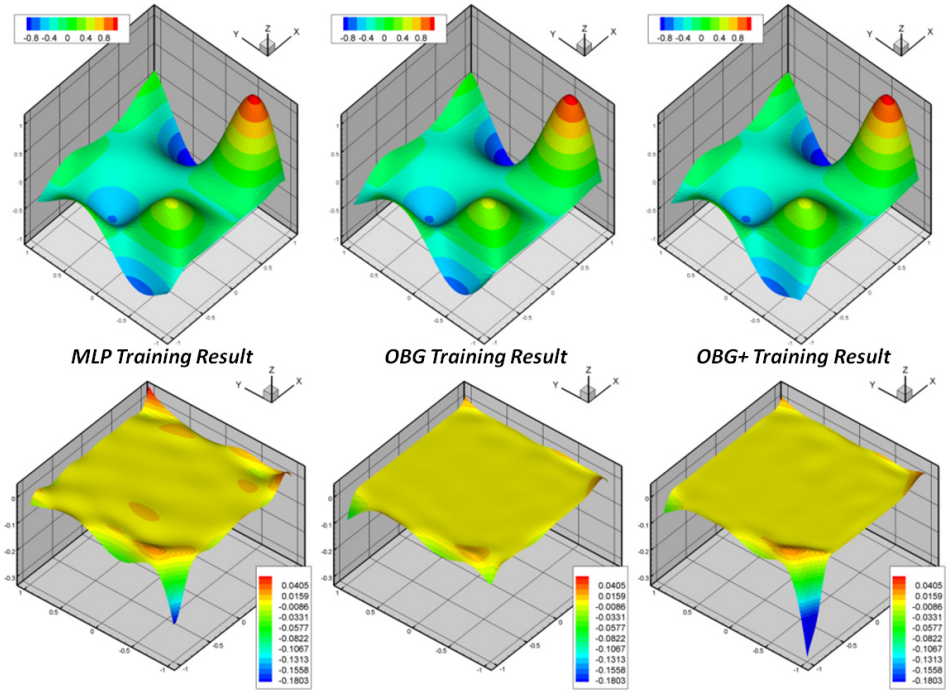


Figure 76: Averaged output of 10 trainings from 2-16-1 network, Experiment 5

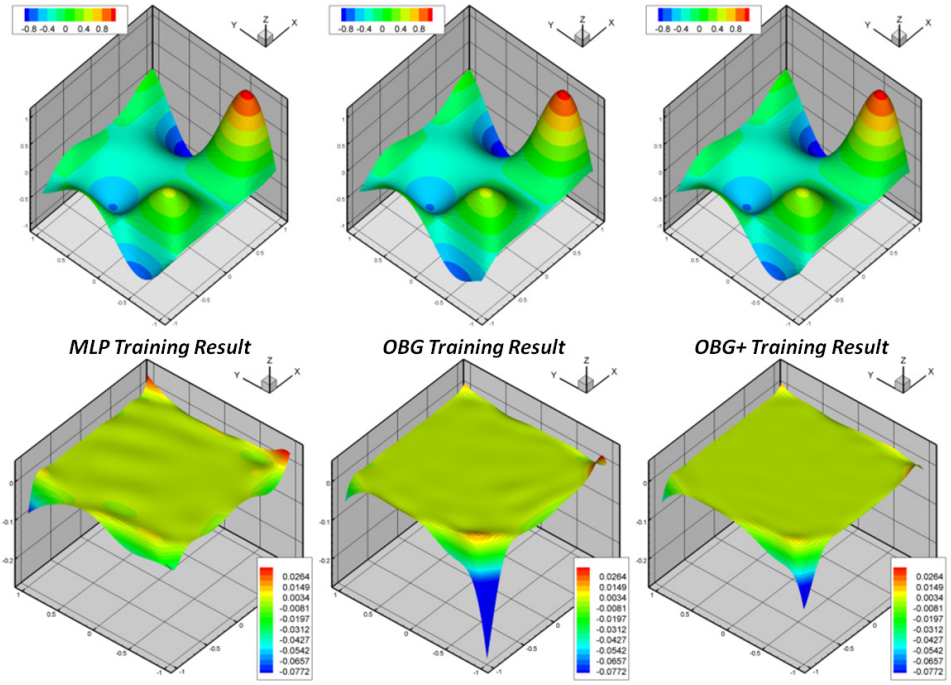


Figure 77: Averaged output of 10 trainings from 2-18-1 network, Experiment 5

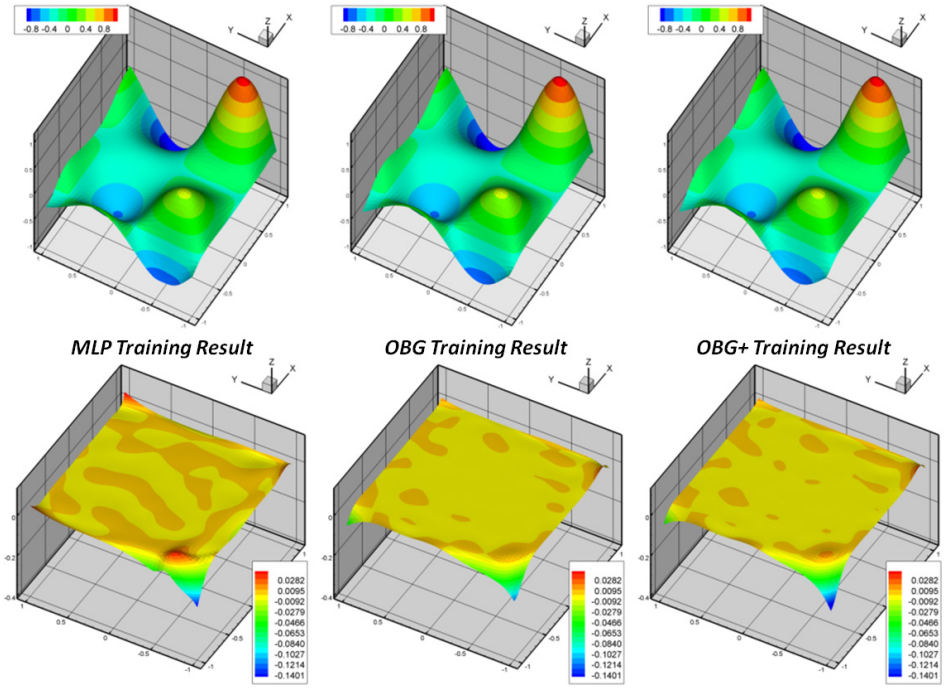


Figure 78: Averaged output of 10 trainings from 2-20-1 network, Experiment 5

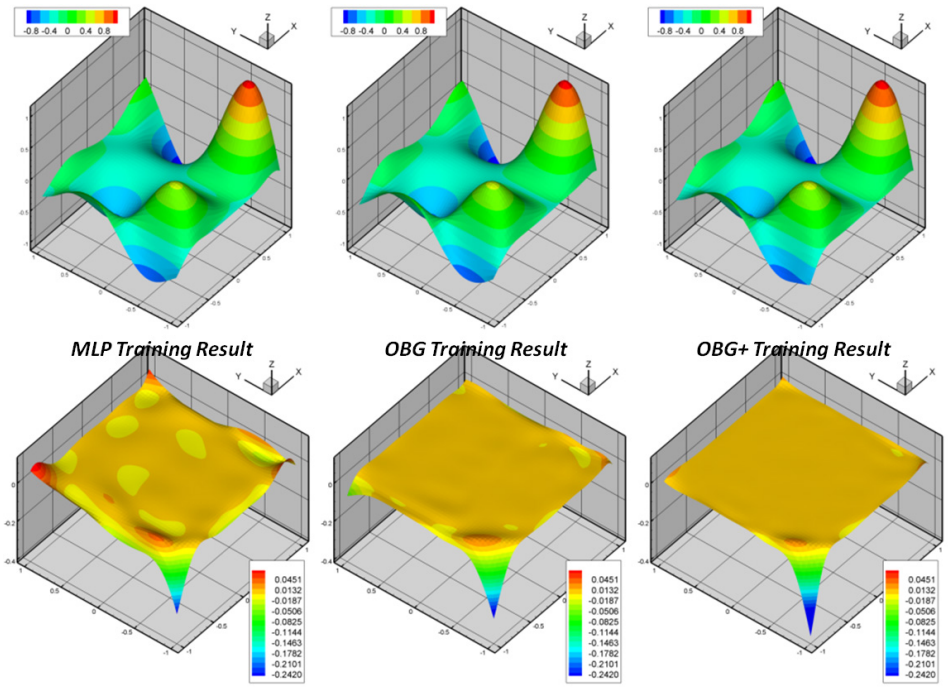


Figure 79: Averaged output of 10 trainings from 2-5-5-1 network, Experiment 5

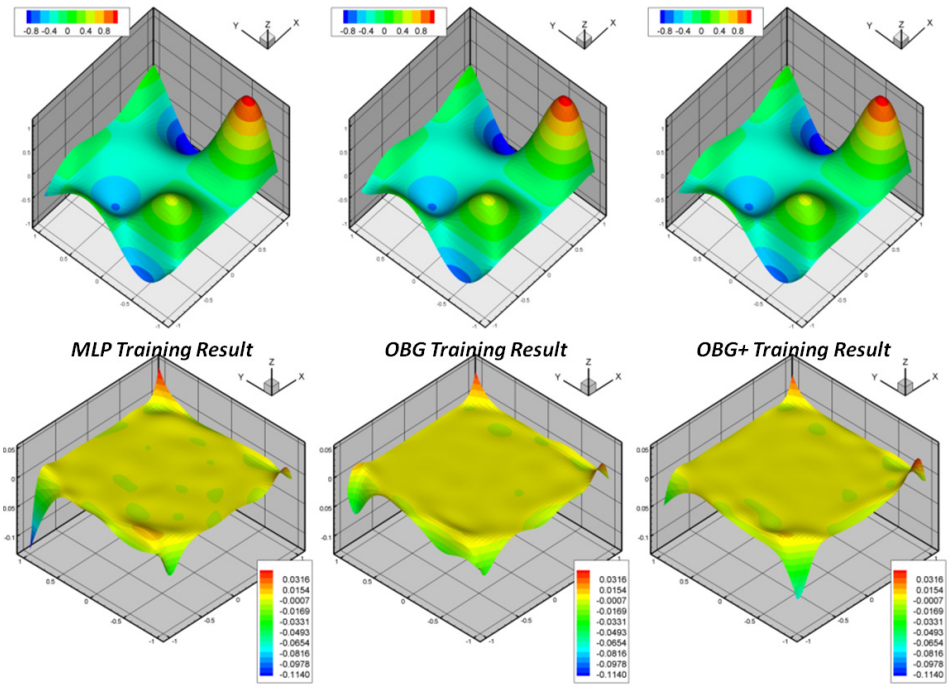


Figure 80: Averaged output of 10 trainings from 2-6-6-1 network, Experiment 5

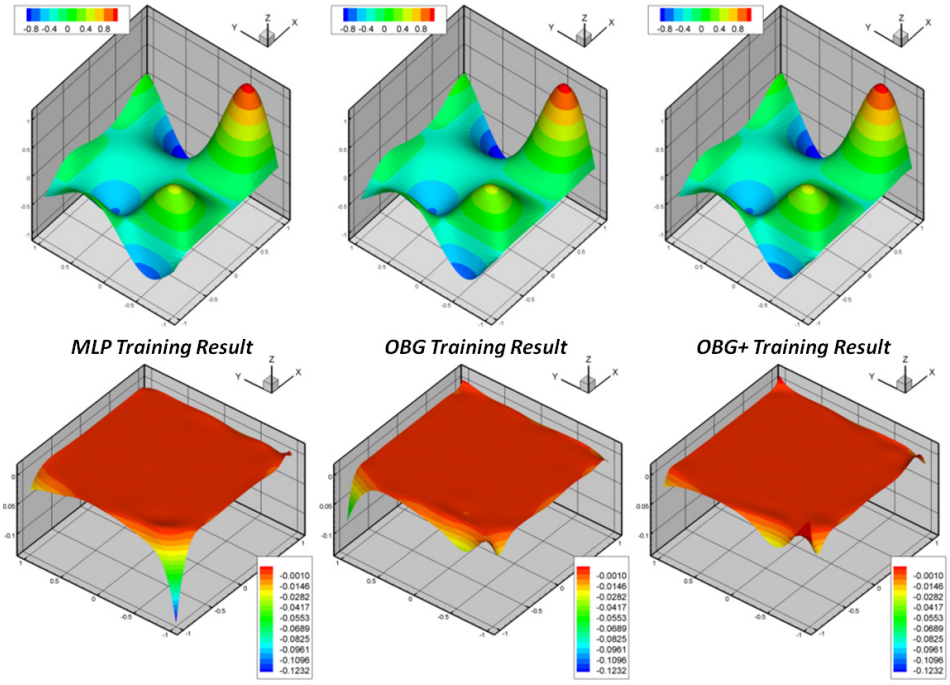


Figure 81: Averaged output of 10 trainings from 2-7-7-1 network, Experiment 5

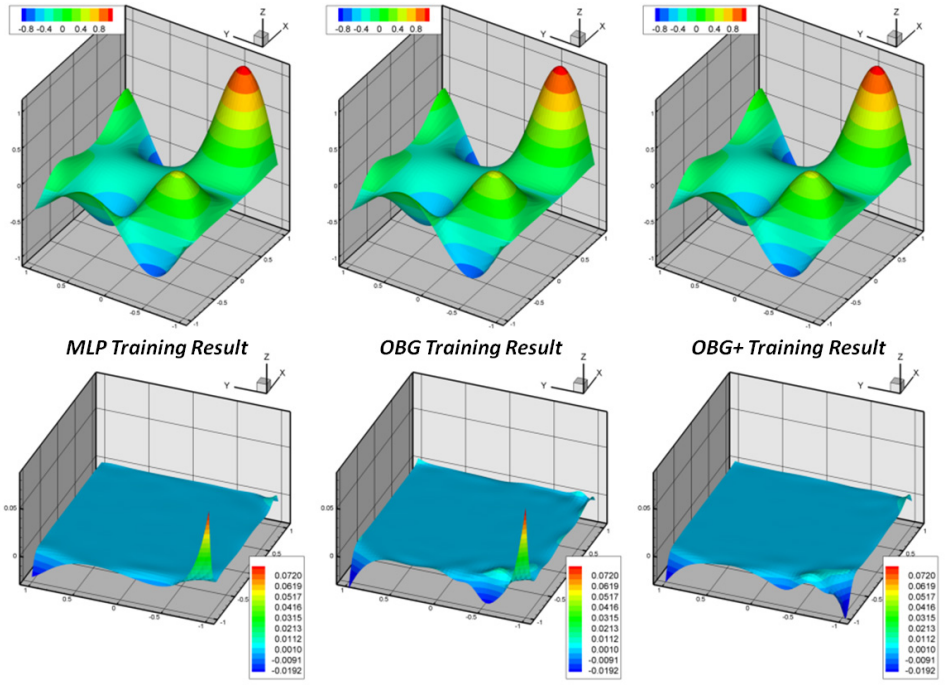


Figure 82: Averaged output of 10 trainings from 2-8-8-1 network, Experiment 5

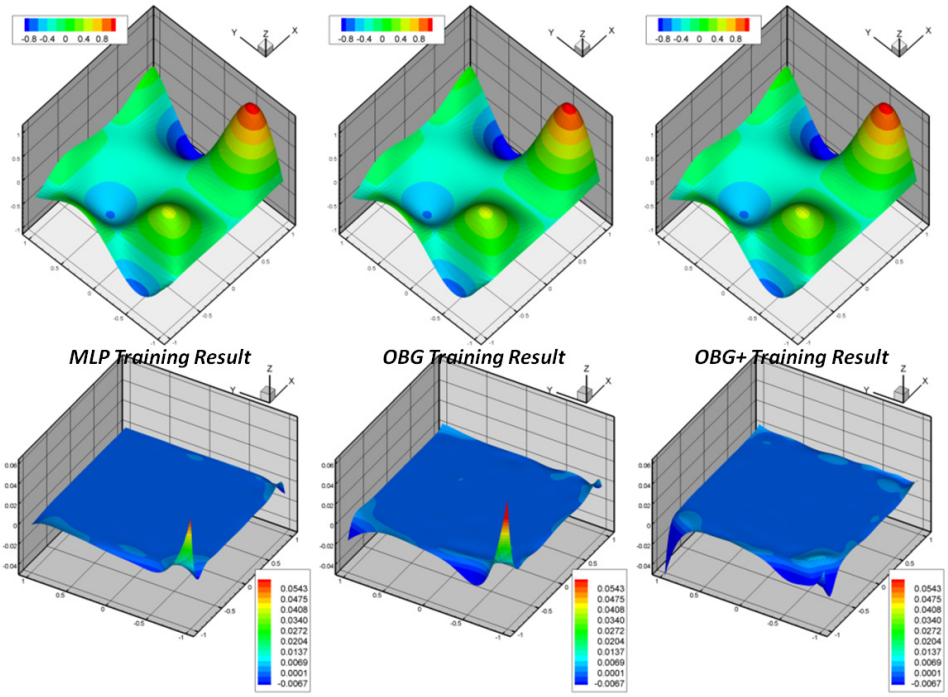


Figure 83: Averaged output of 10 trainings from 2-9-9-1 network, Experiment 5

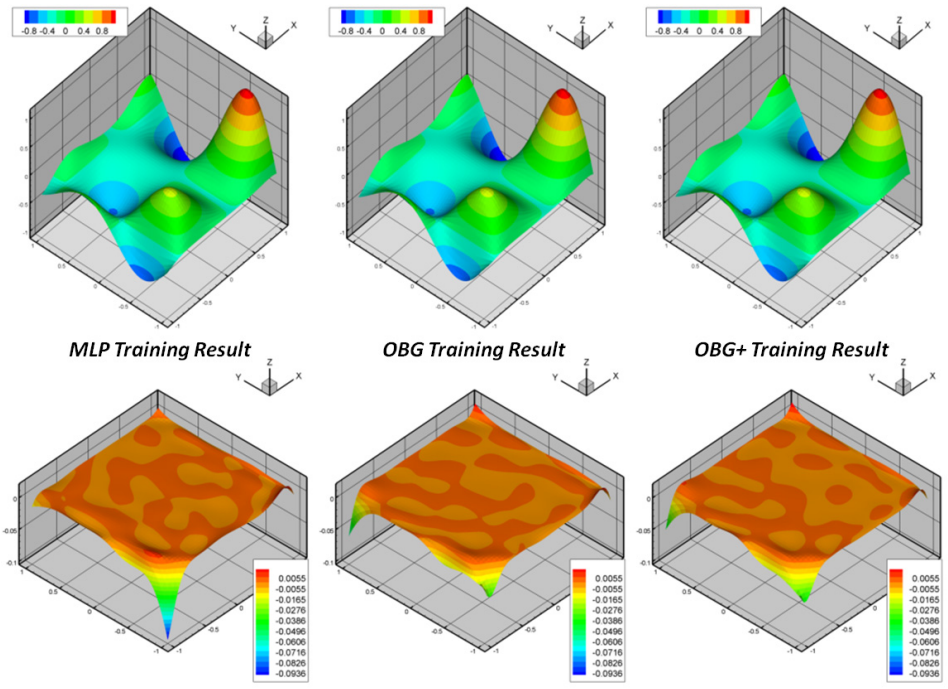


Figure 84: Averaged output of 10 trainings from 2-10-10-1 network, Experiment 5

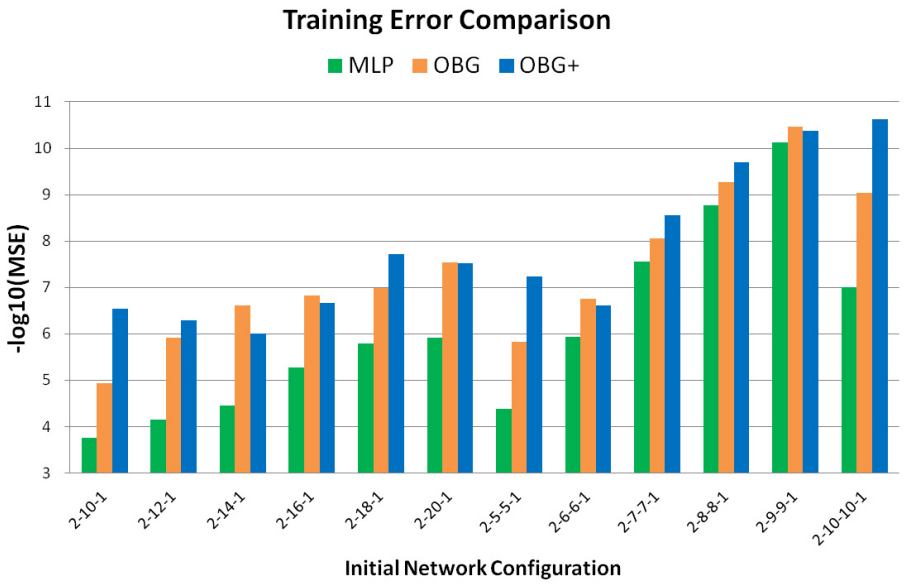


Figure 85: Training error comparison, MLP, OBG and OBG+, Experiment 5

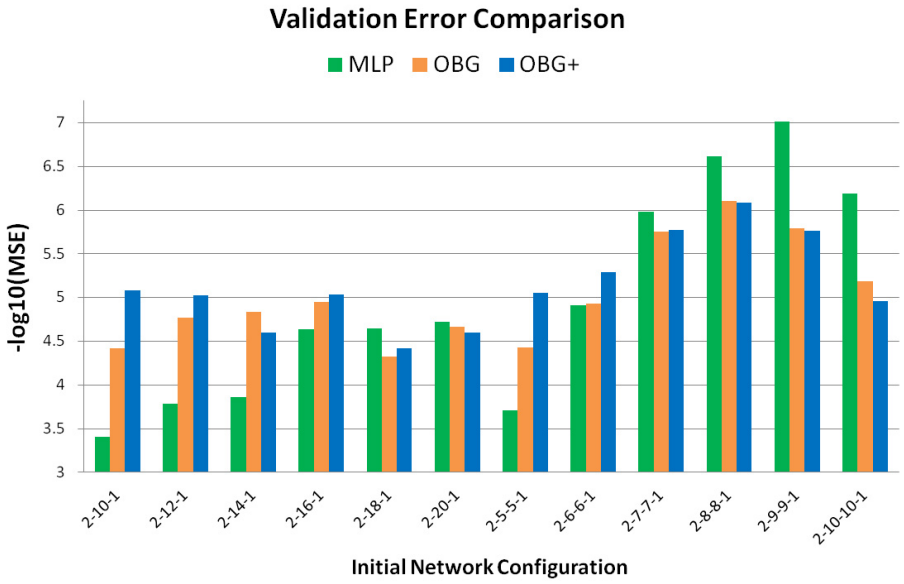


Figure 86: Validation error comparison, MLP, OBG and OBG+, Experiment 5

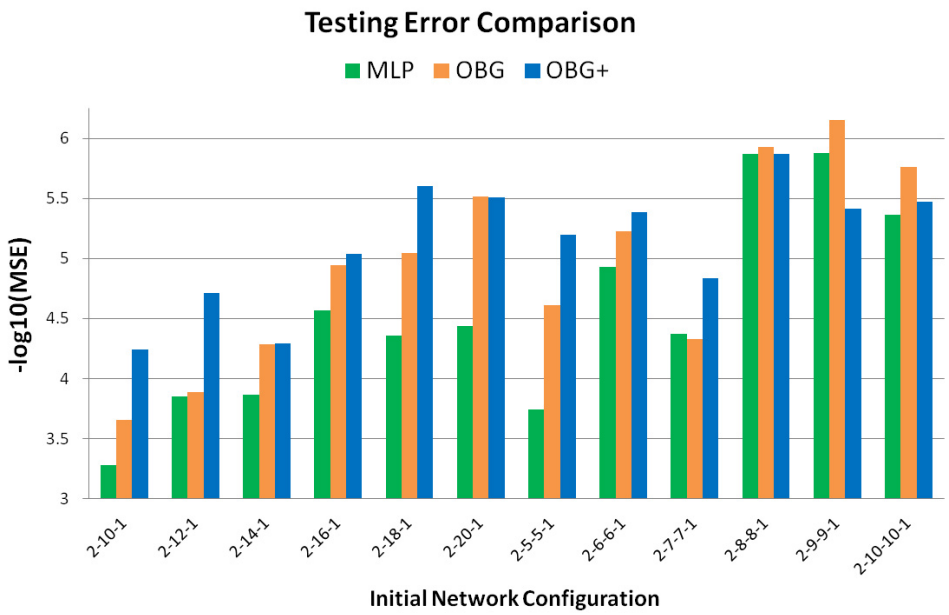


Figure 87: Testing error comparison, MLP, OBG and OBG+, Experiment 5

5.5.3 OBGv+ Scheme Exploiting Validation Information

The consistent result so far show the advantage of the OBG and OBG+ training method in the generalization performance is more evident when the initial network configuration has relatively smaller number of connections and neurons. To maintain this advantage in the training tasks using larger size networks, an exploitation of the validation error has been tried. This approach is based on the observation on the previous experiments including the results of the OBG+ trainings. The efficacy of the growth in the connectivity and the computational units in the network adopted in the OBG and the OBG+ training method has found to be less evident when the initial network is already equipped with the abundant connectivity and computational units such as in the double-hidden layer MLPs having relatively abundant neurons. Therefore, in this case, the growth of connectivity and neurons might have to be suppressed compared to the growth rate tested so far, i.e., one connectivity adjustment per each training epoch. As a criterion to determine the activation and the deactivation of the connectivity adjustment, the deterioration of the validation error has been

chosen, i.e., only when the validation error is increasing the connectivity adjustment is active and, otherwise, the connectivity adjustment is inactive making the training method as same as the conventional BP-LM method. In this way, the OBG and the OBG+ training algorithm is switched to and from the conventional BP-LM method depending on the variation of the validation error. This can be thought as an extension of the role for the validation error monitoring in the conventional MLP training paradigm where its primary role is to determine when the training procedure has to be terminated. In the current case, the role of the validation error monitoring is to determine when the network has to be adjusted in its connectivity and grown with a more computational unit. This scheme has been named as OBGv+ scheme.

To test the generalization performance of the OBGv+ method, the similar system of the training campaign using the exactly same initial conditions has been repeated and compared with the MLP training results.

Table 8 shows the final network status for the MLP and OBGv+ networks where the data for the MLP training results are identical to the results shown in the previous sub-section.

Table 8: Network size of MLP and OBGv+, Experiment 5

Initial Net	N_{neuron}		C_{conn}		$C_{opt} + N_{bias}$	
	MLP	OBGv+	MLP	OBGv+	MLP	OBGv+
2-10-1	13	17.0	30	100.8	41	53.0
2-12-1	15	17.1	36	99.25	49	55.4
2-14-1	17	17.8	42	103.8	57	59.3
2-16-1	19	19.3	48	118.8	65	65.8
2-18-1	21	21.1	54	140.1	73	73.4
2-20-1	23	23.0	60	157.5	81	81.0
2-5-5-1	13	17.8	40	107.4	51	65.3
2-6-6-1	15	18.9	54	122.4	67	78.6
2-7-7-1	17	21.1	70	152.1	85	97.4
2-8-8-1	19	21.8	88	161.3	105	113
2-9-9-1	21	24.0	108	198.3	127	136
2-10-10-1	23	25.9	130	233.0	151	160

The results on the training, validation, and testing errors are provided in Table

9 and Figure 88, 89, and 91. The overall training results show that the generalization performance of the OBGv+ training method brought the obvious enhancement compared to the MLP training cases.

Table 9: Training error comparison for MLP and OBGv+, Experiment 5

Initial Net	Training*		Validation*		Testing*	
	MLP	OBGv+	MLP	OBGv+	MLP	OBGv+
2-10-1	3.77	**4.79**	3.41	**4.11**	3.29	**3.71**
2-12-1	4.15	**5.27**	3.78	**4.59**	3.85	**4.35**
2-14-1	4.47	**5.50**	3.87	**4.48**	3.87	**4.31**
2-16-1	5.28	**6.15**	4.64	**5.35**	4.57	**5.00**
2-18-1	5.80	**6.10**	4.65	**5.38**	4.36	**5.26**
2-20-1	5.91	**6.92**	4.72	**4.84**	4.44	**5.44**
2-5-5-1	4.39	**5.93**	3.71	**4.77**	3.75	**4.84**
2-6-6-1	5.94	**7.08**	4.91	**5.85**	4.93	**5.78**
2-7-7-1	7.57	**8.13**	**5.98**	5.95	4.37	**4.57**
2-8-8-1	8.77	**9.21**	6.62	**6.68**	5.87	**6.09**
2-9-9-1	10.1	**10.3**	**7.01**	7.00	5.88	**6.64**
2-10-10-1	7.00	**9.81**	6.19	**7.11**	5.37	**6.39**

*Errors in $-\log_{10}(MSE)$; the bigger, the less error.

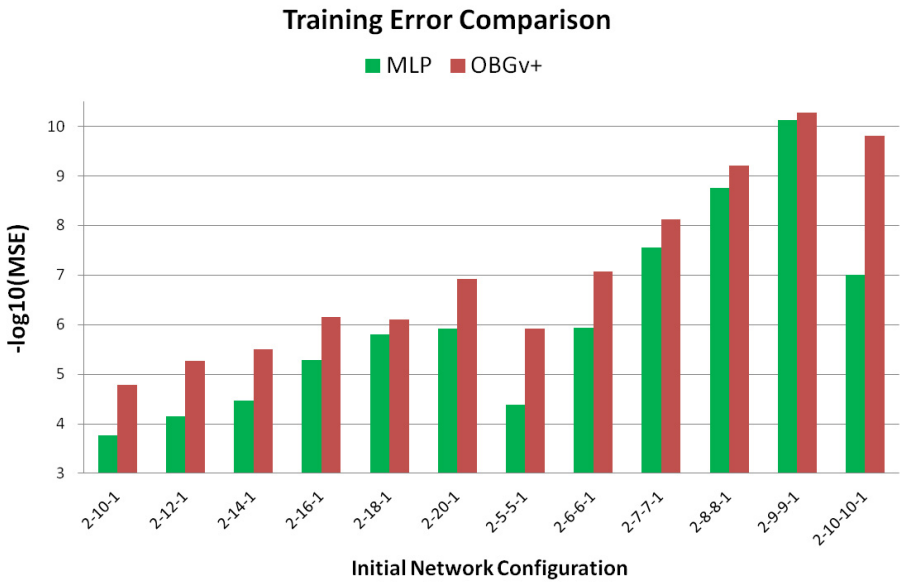


Figure 88: Training error comparison, MLP and OBGv+, Experiment 5

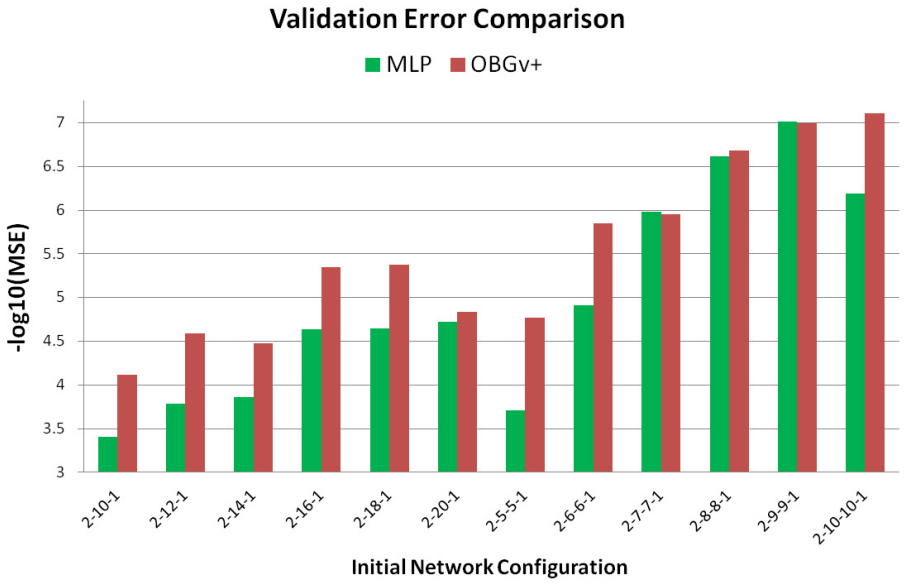


Figure 89: Validation error comparison, MLP and OBGv+, Experiment 5

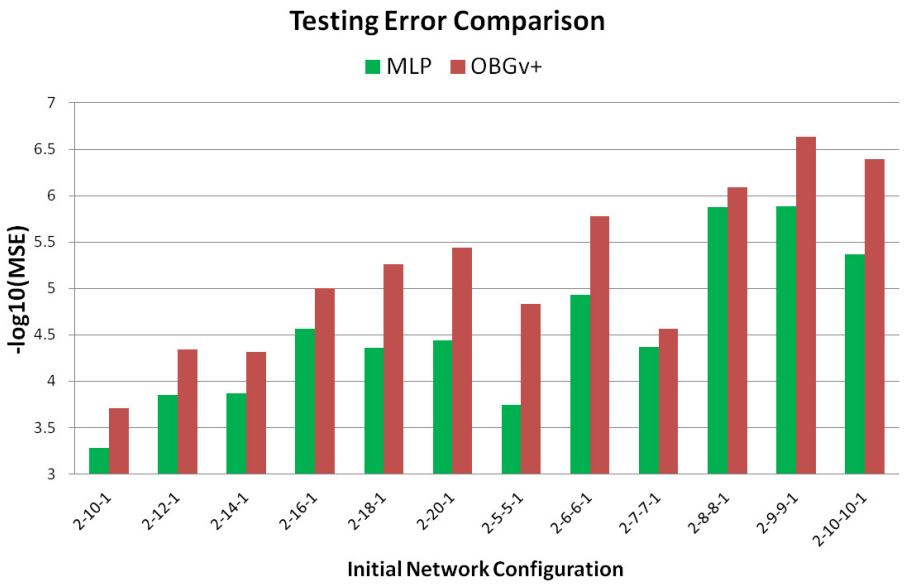


Figure 90: Testing error comparison, MLP and OBGv+, Experiment 5

Table 10: Training time comparison for MLP and OBGv+, Experiment 5

Initial	Epoch*		Second*		Second/Epoch	
Net	MLP	OBGv+	MLP	OBGv+	MLP	OBGv+
2-10-1	6498	5455	13.0	16.2	0.0020	0.0030
2-12-1	1280	6846	3.42	24.1	0.0027	0.0035
2-14-1	7969	5546	29.2	21.5	0.0037	0.0039
2-16-1	7012	3641	30.9	17.2	0.0044	0.0047
2-18-1	9302	6559	54.8	40.7	0.0059	0.0062
2-20-1	4032	11542	28.3	84.4	0.0070	0.0073
2-5-5-1	5077	11514	13.4	46.8	0.0026	0.0041
2-6-6-1	4039	13269	17.0	78.8	0.0042	0.0059
2-7-7-1	14827	13637	107	124	0.0072	0.0091
2-8-8-1	13292	16233	153	213	0.0115	0.0131
2-9-9-1	16937	17732	302	362	0.0178	0.0204
2-10-10-1	12437	15301	341	454	0.0274	0.0297

*Number of seconds and epochs to the minimum validation error.

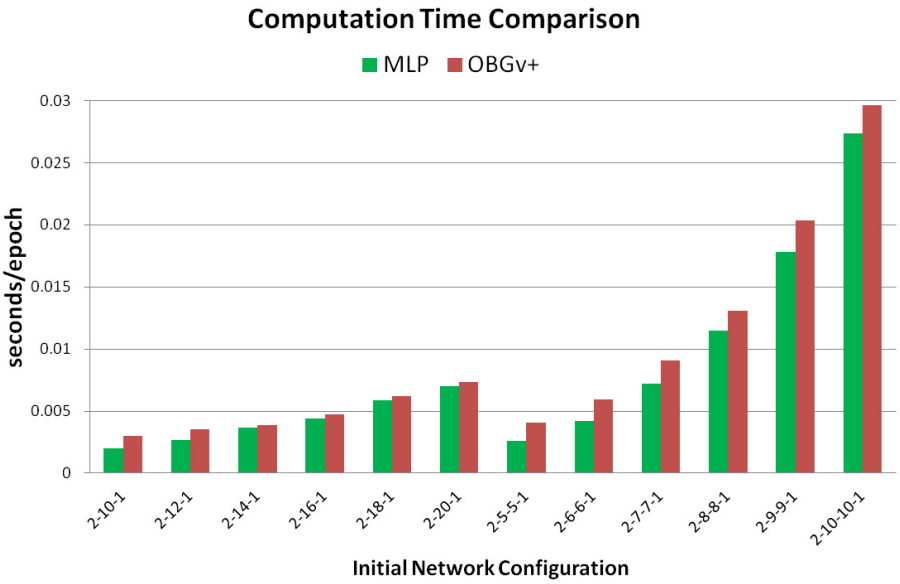


Figure 91: Computation time comparison, MLP and OBGv+, Experiment 5

CHAPTER VI

TRANSONIC AIRFOIL MODELING

In this chapter, the modeling capability of the developed connectivity adjusting learning scheme, the Optimal Brain Growth algorithm, for the practical surrogate modeling of the CFD analysis is described. As a benchmarking test, the surrogate modeling task has been performed for the aerodynamic characteristics of two-dimensional airfoils in the transonic flow regime. Contrast to the more straightforward cases of subsonic airfoil modeling via neural networks, the transonic CFD analysis requires significantly more training effort to be properly surrogated via conventional neural network techniques such as the BP training method and the BP-LM method using the MLP networks. This challenge in the mathematical modeling in the transonic flow regime is primarily due to the existence of the complicated aerodynamic phenomena involving discontinuities such as the shock waves as shown clearly in the previously described literature survey. The current application of the OBG training method and the comparative study to the more conventional BP-LM training using the MLP networks is an effort to widen the applicability of the neural network technique in the transonic flow regime by enhancing network training efficiency. To perform an appropriate modeling, not only the training method but also the representation of the input and output parameters have to be efficiently designed. For the generation of target data, one of the simplest CFD methods has been chosen considering the balance between the physical accuracy of the flow analysis and the numerical efficiency in its implementation. Using the CFD analysis, training examples consisting of input data defining the flow condition and the geometric properties of the airfoil and corresponding aerodynamic analysis results represented by surface pressure coefficients on

the various positions have been generated. Then, the surrogate modeling has been performed using the conventional BP-LM method and the OBG training method to investigate the difference in their modeling capability and efficiency.

6.1 CFD Analysis using TSFOIL

As the Mach number of free-stream flow increases from the subsonic one, it reaches at a particular Mach number in which the local flow becomes a sonic speed at a single point on the surface. This particular free-stream Mach number is the *critical* Mach number. As the free-stream Mach number increases further, a finite region of supersonic flow is developed through which the flow speed is slowed to subsonic one via the shock wave. This general development of the shock wave is illustrated in Figure 92 [57].

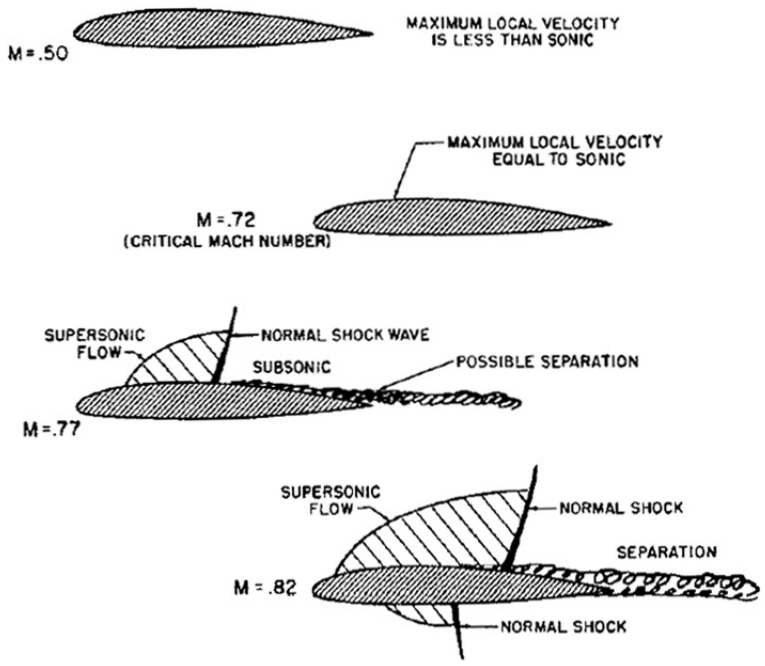


Figure 92: Progression of shock waves with increasing Mach number [57]

Following the Mason's description [57], just after *conquering* the blunt body problem in 1970s, *everybody* was trying to devise a numerical method to analyze this phenomenon, i.e., the transonic flow over an airfoil. The challenging difficulty to achieve this goal was the inherent nonlinearity of the phenomenon and the coexistence of dissimilar types of governing equations, i.e., being elliptic in the subsonic region and hyperbolic in the supersonic region of the flow even in the steady flow conditions, which can be expressed in the small disturbance theory as;

$$\left[1 - M_\infty^2 - (\gamma + 1)\, M_\infty^2 \phi_x\right] \phi_{xx} + \phi_{yy} = 0 \tag{82}$$

where, depending on the sign of the bracketed term, the type of the equation becomes elliptic partial differential equation (when it is positive) or hyperbolic one (when it is negative). Using the finite difference approximation, Murman and Cole tackled this *switching* nature in the governing equation by *switching* the form of the differencing term resulting in "mixed differencing" scheme [79]. The essence of this scheme is to estimate the flow at each calculation point to determine the flow is subsonic or supersonic and, then, a central difference is used for the second derivative in the x-direction in case of subsonic flow and an upwind difference is used for supersonic flow. In this process, the shock wave naturally emerges in the solution. This scheme was the first practical *shock-capturing* method making its extension to the general three-dimensional analysis much simpler contrast to the competing *shock-fitting* methods in which shocks had to be located and the Rankine-Hugoniot conditions were applied. This scheme has been regarded as the major breakthrough in the analysis of transonic airfoil and implemented in the code known as TSFOIL, which is the final development of small disturbance theory for two-dimensional flow analysis [57]. Jameson, after hearing the presentation by Murman in 1970, further developed the scheme resulting in the efficient full potential flow code known as FLO36. Mason summarized the successive development in the field like this [57]; *"The next logical development was to add viscous effects to the inviscid calculations, and to switch to the Euler equations*

for the outer inviscid flow. By now, many researchers were working on computational flow methodology, which had become an entire field known as CFD." Therefore, among the various CFD methods available today, the code TSFOIL is positioned in the unique position as the oldest and the simplest one which can successfully capture the essence of the transonic flow around the airfoil including shock waves.

In the current transonic airfoil modeling, the code TSFOIL2 which has been modified from the original TSFOIL is used to generate target pressure distributions. One important factor for this choice is that TSFOIL calculates pressure distribution around airfoil using only the surface coordinates of the airfoil not requiring the computational grid externally wrapping the airfoil, which is a great advantage in obtaining flow solution efficiently, eliminating the need for effort of grid generation and the uncertainty originating from the quality of the grid system.

There exist issues related to the quality of the flow analysis results such as the accuracy of the estimated pressure distribution and the strength and the location of the shock wave in case of its occurring. But, in the current thesis, the main concern is the accuracy and efficiency of the surrogate modeling rather than the accuracy of the CFD analysis itself. In other words, as long as the CFD analysis is able to capture the fundamental characteristics of the flow around the airfoil shapes of interest, the numerical accuracy of the CFD analysis itself is only a secondary issue. Figure 93 provides one example comparing the analysis qualities from the three different CFD codes; TSFOIL2, FLO36, and MSES [57]. Mason pointed out two primary reasons of *inexactness* of two other solutions compared to the more *exact* Euler solution; First, the correct shock jump and, second, the loss in stagnation pressure across the shock wave are not dealt with in the potential flow model.

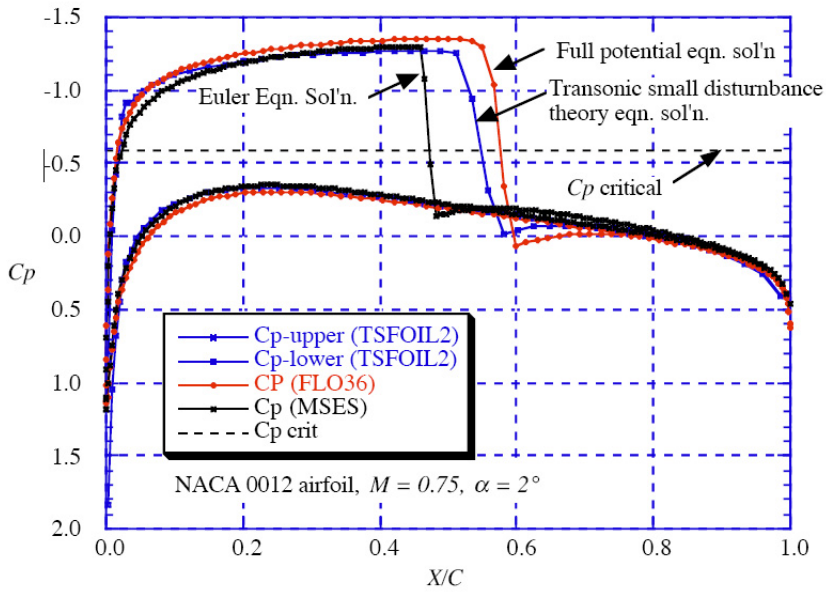


Figure 93: Comparison of pressure distribution on an NACA 0012 airfoil at Mach 0.75, AoA 2^o using TSFOIL2, FLO36, and MSES [57]

6.2 Modeling Scope

The purpose of the current modeling is to probe the applicability of a novel surrogate modeling technique for the practical surrogate modeling problem. Within this context, the scope of modeling for the transonic airfoil analysis has been determined as a moderate one rather than a complete and definitive model covering wider range of airfoil shapes and flow conditions. The basic framework of the modeling is to use following input and output parameters;

- Input parameters
  - Flow conditions represented by the free-stream Mach number and the incidence of airfoil (or angle of attack)
  - Airfoil geometric parameters
- Output parameters
  - Surface pressure coefficients of multiple points on the airfoil, which can

approximate entire pressure distribution of the airfoil including the upper and lower surfaces

Among these parameters, the way each airfoil is geometrically represented also affects the representation of the pressure distribution. Assuming the number of shape parameters defining the entire geometry of airfoil and the required number of the pressure measurement points are not so small, the determination of the method for geometric representation of the airfoil is an important issue affecting the overall network design. This determination is also governed by a non-trivial decision for one of the following possible two strategies;

- To use the position of pressure prediction as an input parameter of the model; In this case, the number of output nodes decreased to only one corresponding to the pressure coefficient of the position defined by the input parameter but the required training examples have to be significantly extended to discriminate each and every position's pressure coefficient for fundamentally identical input parameter set which is only distinctive by the position parameter.

- Not to use the position of pressure prediction and to use multiple output nodes to represent entire pressure distribution; In this case, the network size increases due to the multiple output nodes but the training data becomes more concise.

Assuming the second-order optimizer adopting *batch* training where the unit training in each epoch is executed using the entire set of training examples, the size of the Hessian matrix depends on the term, the number of the training examples multiplied by the number of the output nodes. Therefore, either of above two strategies where the value of this term is conserved has no obvious advantage in the memory consumption during the training but the absolute number of processes for the forward input signal propagations and the backward error propagations is significantly increased in

case of choosing the first strategy simply because the instance of the training examples are increased. Therefore, to promote the training efficiency at the expense of the flexibility in the modeling capability allowing prediction on the arbitrary geometric location, the second strategy has been chosen for the current study. Beyond the aspect of the required training resources, there exists a more fundamental issue on the qualitative difference between above two strategies such as the possible advantage in case of adopting the second strategy by the network being concurrently trained using all relevant outputs, which is conceptually in the opposite side to the *neural interference* or *cross-talk* where the network is confused by being trained simultaneously for dissimilar tasks [88]. In case of aerodynamic coefficient modeling using the neural network, two separated networks for lift coefficient and drag coefficient are known to result in the better modeling capability than the single network trained simultaneously for the two outputs, which correspond to the aerodynamic version of the *neural interference*. At present, this issue is beyond the scope of current thesis partly due to the obvious lack of relevant research contents in the literature.

6.3 Representation of Airfoil Geometry

The most widely-used methods for representing airfoil geometry can be listed as follows;

- Direct assignment of airfoil coordinates for each and every discretized points on the upper and the lower surface
- Defining shape function with its parameters and linearly combining multiple shape functions
- Spline-type methods via assignment of the control points, which is usually less than the discretized points on the airfoil
- Parametric representation from which airfoil coordinates are generated

Recently, a particular parametric method known as *PARSEC* method [30] is gaining popularity in the geometric representation of transonic airfoils [46], [64], [87], [78], and [85] due to its efficiency and flexibility. The PARSEC method was developed by Sobieczky aiming the efficient and flexible geometric representation using 11 geometric parameters particularly chosen for the *modern* airfoils. Figure 94 shows the schematic diagram indicating these 11 parameters and some examples of represented airfoil shapes.

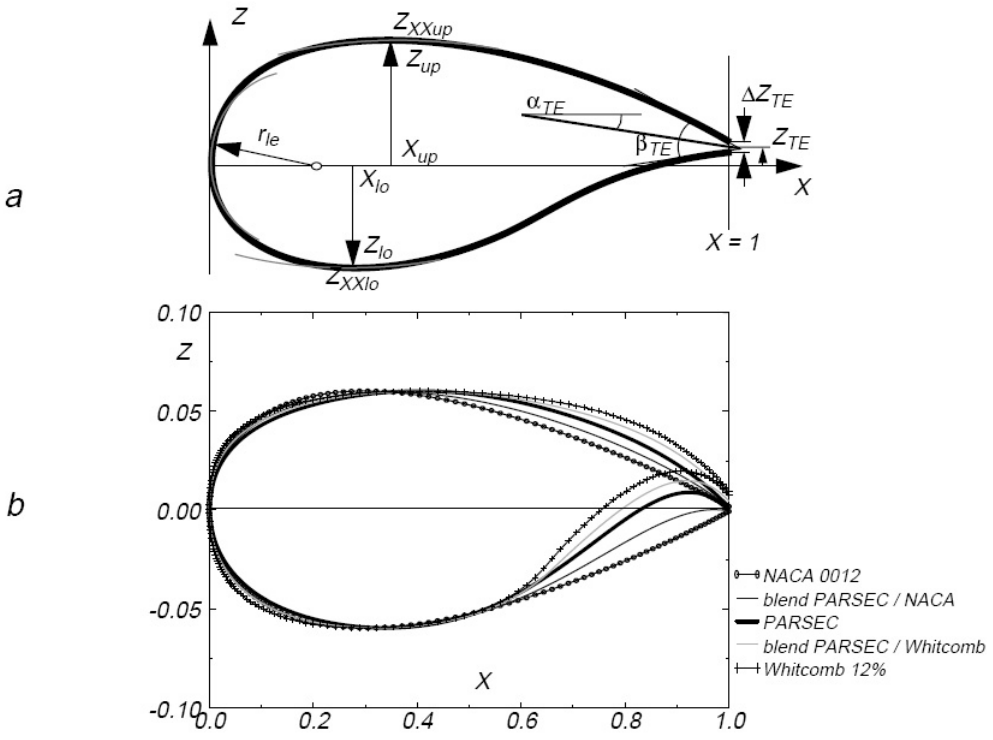


Figure 94: Geometric parameters for PARSEC airfoil and its variation by blending with NACA or Whitcomb airfoil

For the current study, the PARSEC method has been used to represent airfoil geometry excluding the parameter Z_{TE} and δZ_{TE}. The parameter Z_{TE} represents the vertical disposition of the training edge point and the effect of this parameter is already included in the airfoil incidence information. The impact of parameter δZ_{TE} which represents the finite thickness at the training edge is simply neglected in the current study considering the capability of analysis code which is for fundamentally

inviscid flow. Therefore, total 9 geometric parameters are used to completely define the airfoil shape. Straightforward interpretation of these geometric parameters results in the following constraining equations;

$$Z(1) = Z_{TE} = 0 \tag{83}$$

$$Z(X_{UP}) = Z_{UP} \tag{84}$$

$$Z'(1) = tan(\alpha_{TE} + \frac{1}{2}\beta_{TE}) \tag{85}$$

$$Z''(X_{UP}) = Z_{XX_{UP}} \tag{86}$$

$$Z(\epsilon) = \sqrt{2 * R_{LE}\epsilon - \epsilon^2} \tag{87}$$

$$Z'(Z_{UP}) = 0 \tag{88}$$

where ϵ can be assigned as any very small real number to define the circular portion of airfoil near leading edge. These 6 equations can be solved using the polynomial for the airfoil shape which has 6 unknown coefficients;

$$Z = \sum_{n=1}^{6} a_n X^{n-\frac{1}{2}} \tag{89}$$

This procedure solving 6 linear equations needs to be executed both for the upper and the lower airfoil surfaces independently using the corresponding geometric parameters. Referring the research papers on the transonic airfoil optimization using PARSEC method for airfoil geometric representation, following ranges for adopted 9 PARSEC parameters have been determined around the classical high-performing transonic airfoil RAE2822 [87].

Table 11: Range of PARSEC parameters for airfoil geometry

Parameter	Description	Min.	RAE 2822	Max.
R_{LE}	Leading edge radius	0.002	0.0085	0.02
X_{UP}	Upper crest abscissa	0.35	0.4324	0.5
Z_{UP}	Upper crest ordinate	0.04	0.063	0.1
$Z_{XX_{UP}}$	Upper crest curvature	-0.8	-0.4363	-0.2
X_{LO}	Lower crest abscissa	0.3	0.3438	0.5
Z_{LO}	Lower crest ordinate	-0.06	-0.0589	-0.02
$Z_{XX_{LO}}$	Lower crest curvature	0.3	0.7006	0.9
α_{TE}	Training edge direction	-10.0	6.81	10.0
β_{TE}	Training edge wedge angle	4.0	8.08	12.0

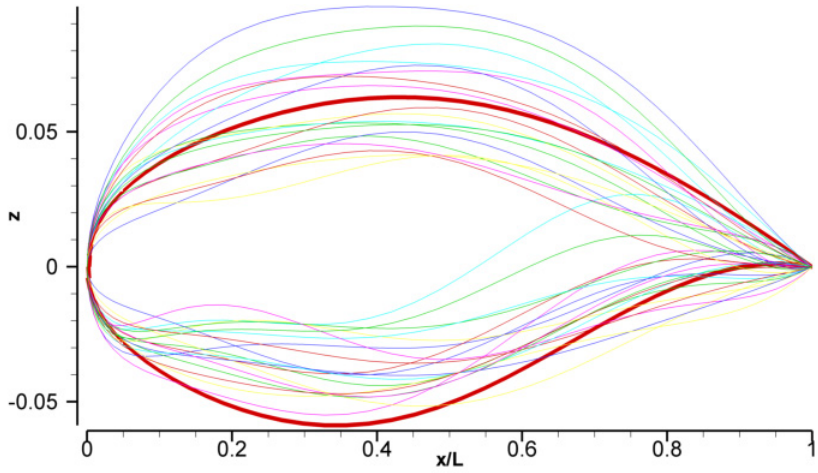


Figure 95: 20 airfoil samples constructed from random selection for 9 PARSEC parameters within the defined range and RAE 2822 airfoil (with thicker line)

6.4 Geometry-Pressure Mapping in Subsonic Flow Regime

Before generation of surrogate model for the transonic flow regime, the low Mach number modeling has been performed to check the validity of the modeling plan. As reported by Hazarika et al. [38], the pressure distribution over airfoils in the subsonic flow regime can be successfully modeled by the conventional MLP networks. In the current study, two independent networks are trained for the upper surface and the lower surface of the airfoils considering the multitude of the pressure probing positions. In the subsonic cases, each surface has been discretized by 15 points. Using the TSFOIL2 code, the training data set of total $2,000$ training examples has been constructed. Mach number range is from 0.5 to 0.55 and the angle of attack varies from negative 2 degrees to positive 4 degrees. Combined with the 9 geometric parameters defined in the previous section, total 11 input parameters are randomly chosen to generate this $2,000$ training examples. Therefore, each network has 11 input nodes and 15 output nodes. The number of hidden units and the initial network structure have been determined as the single-hidden layer MLP having 12 hidden neurons considering the previous modeling case by Hazarika et al. [38]. Figure 96 shows the variation of training, validation, and testing error for the training of upper surface pressure distribution. The OBG training converges towards the lower error for all three types of errors. Figure 97 is a diagram for the OBG trained network. Figure 98 and Figure 99 are the corresponding plots for the training of lower surface pressure distribution. Figure 100, Figure 101, and Figure 102 provide randomly selected examples of the pressure distribution comparison with the target data for the MLP and the OBG trained networks. Here, the orange symbols indicate the prediction from the OBG trained networks and the green symbols indicate the prediction from the MLP networks. All examples from training set (Figure 100), validation set (Figure 101), and testing set (Figure 102) show the reasonable agreement to the target pressure distribution.

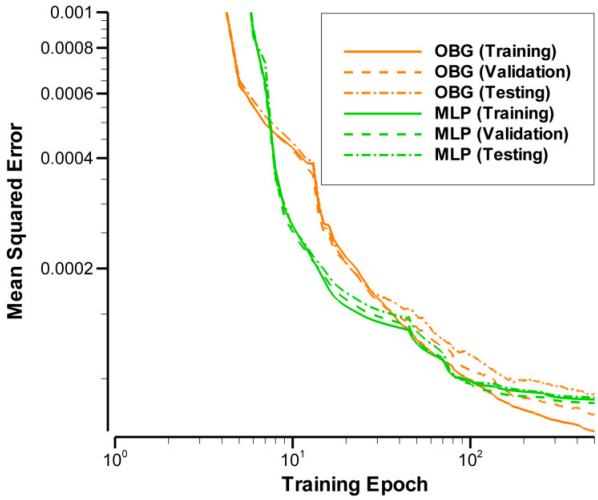


Figure 96: Error variation in subsonic modeling (upper surface network)

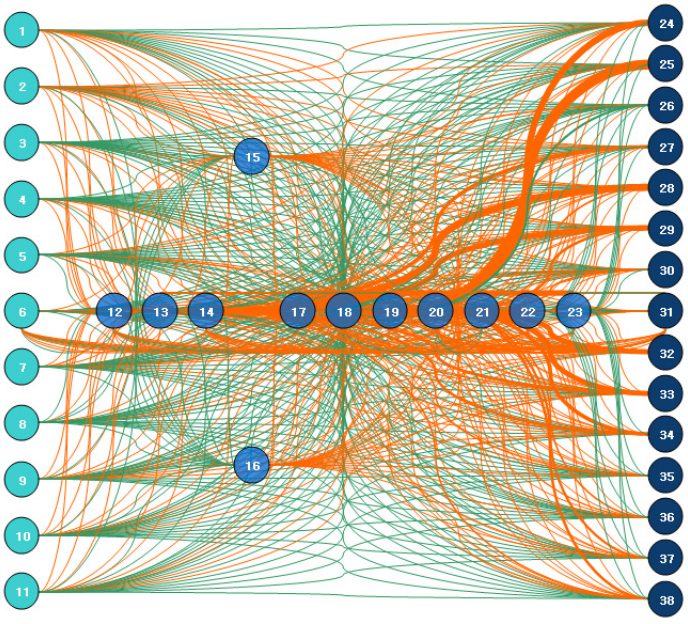


Figure 97: OBG trained network in subsonic modeling (upper surface network)

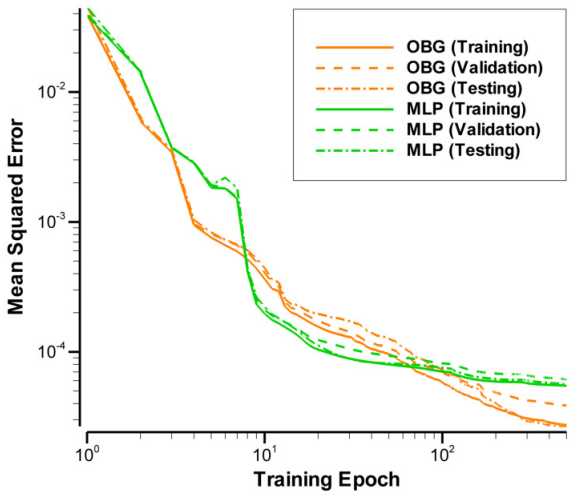


Figure 98: Error variation in subsonic modeling (lower surface network)

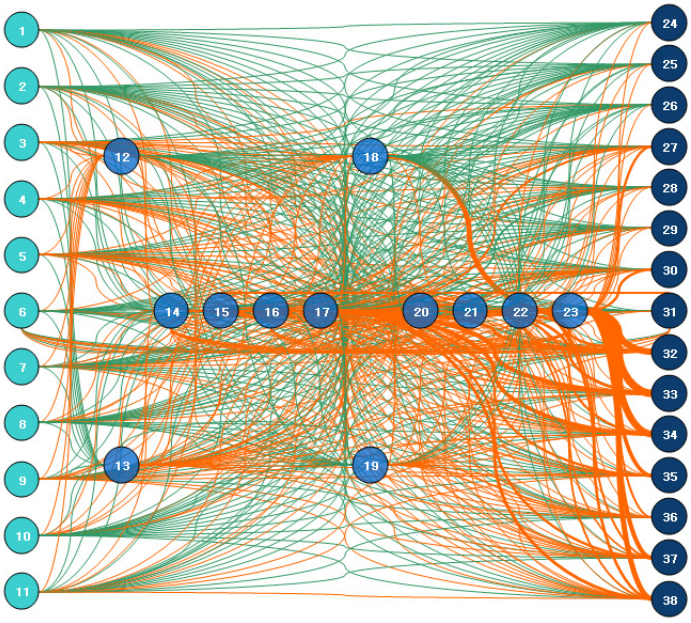


Figure 99: OBG trained network in subsonic modeling (lower surface network)

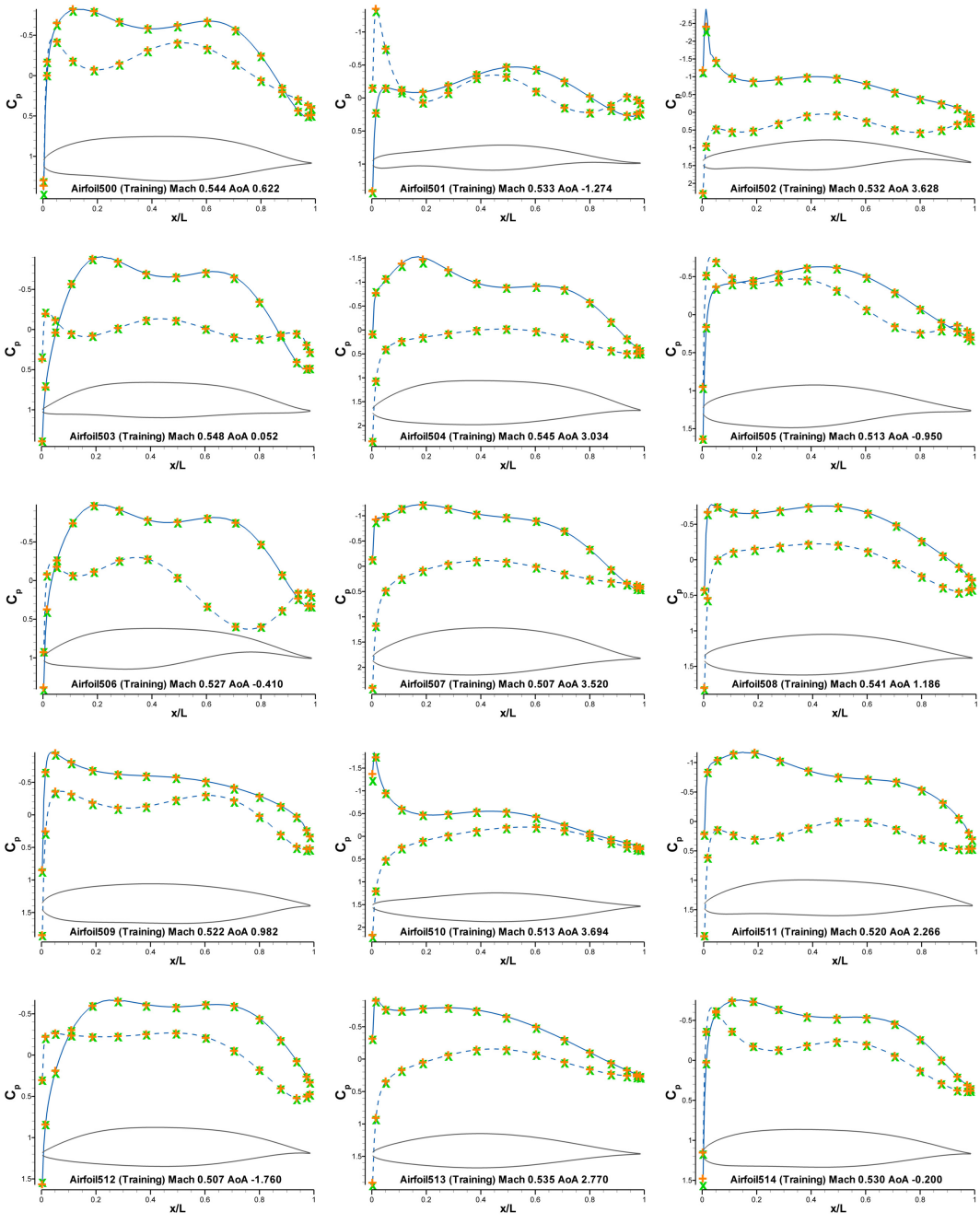


Figure 100: Subsonic airfoil modeling result (training samples)

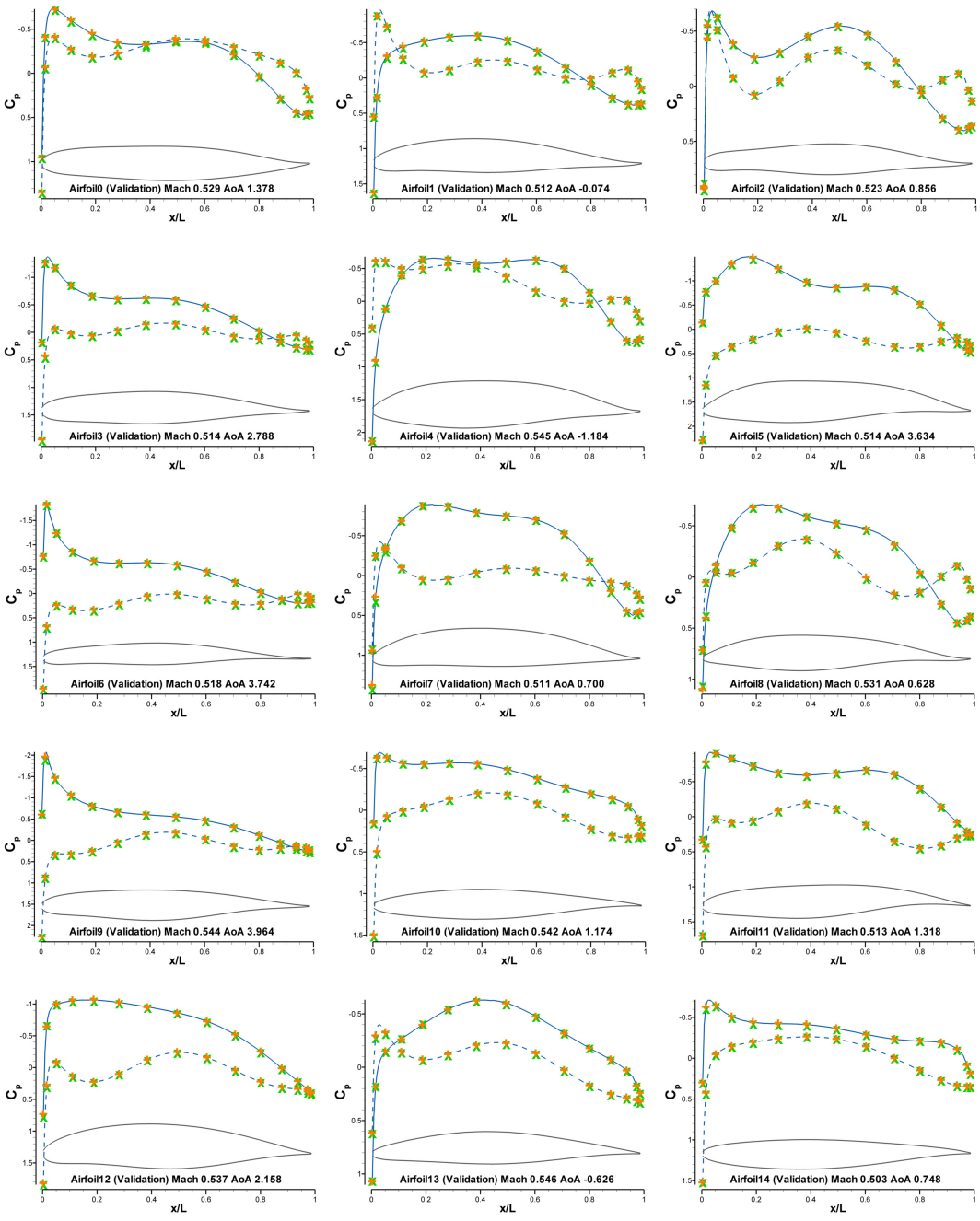


Figure 101: Subsonic airfoil modeling result (validation samples)

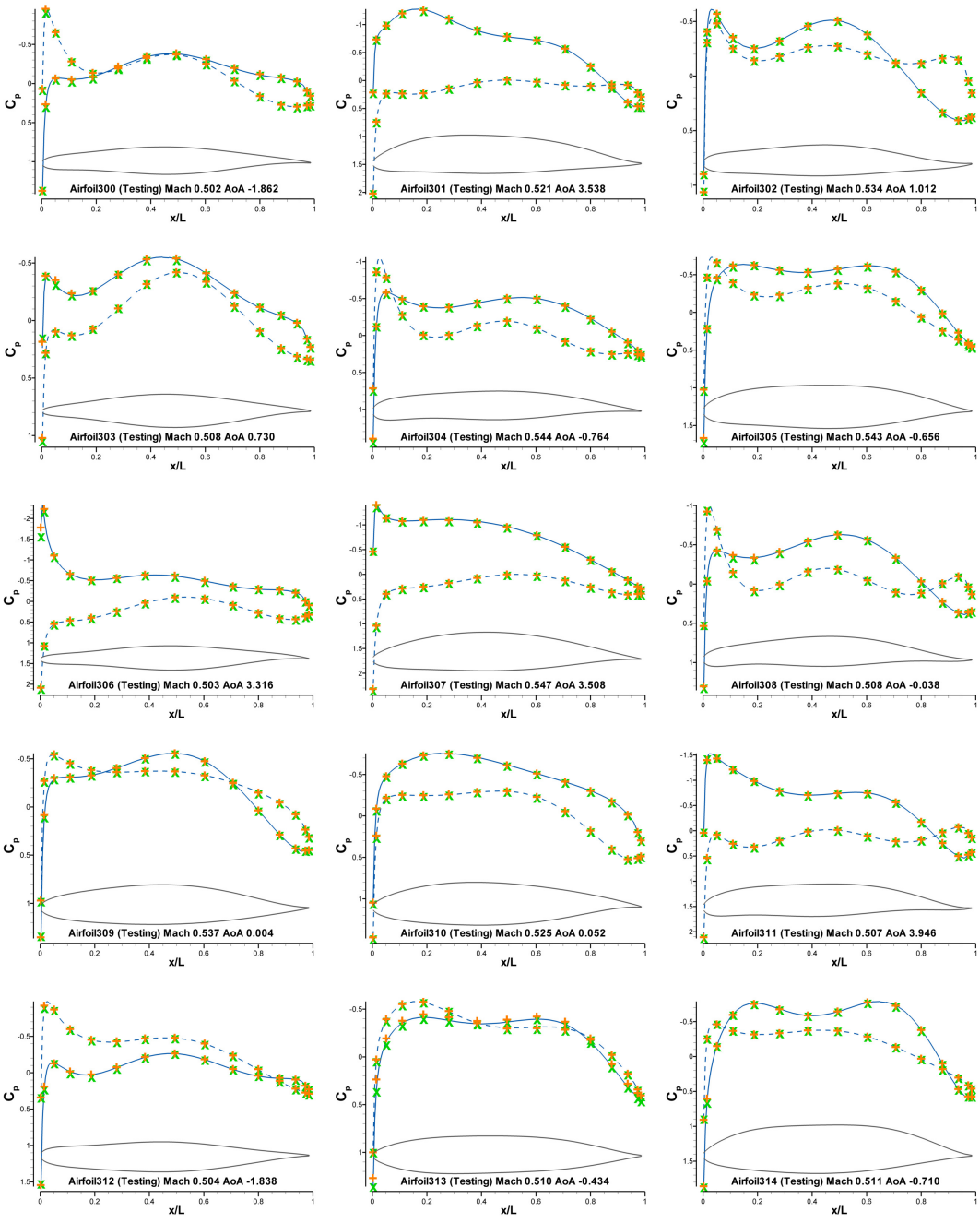


Figure 102: Subsonic airfoil modeling result (testing samples)

6.5 Geometry-Pressure Mapping in Transonic Flow Regime

For the pressure distribution modeling in the transonic flow regime, total 1,629 training examples have been obtained by TSFOIL2 analyses for the Mach number ranging from 0.65 to 0.75 and the angles of attack from -2 to 4 degrees. Considering the larger variability in the pressure distribution compared to the subsonic cases, each network for the upper or the lower surface has been trained for the 20 pressure probing points on the airfoil. Therefore, in this modeling, each network has 11 input nodes and 20 output nodes.

6.5.1 Lower Surface Models

For the modeling of the pressure distribution for the lower airfoil surface, the OBG+ scheme has been used to train the double-hidden layer MLP network having 10 hidden neurons in each layer. Figure 103 shows the variation of the training, validation, and testing errors both for the OBG+ training and the BP-LM training using MLP network. Similar trend in all three types of error is evident as the subsonic modeling case.

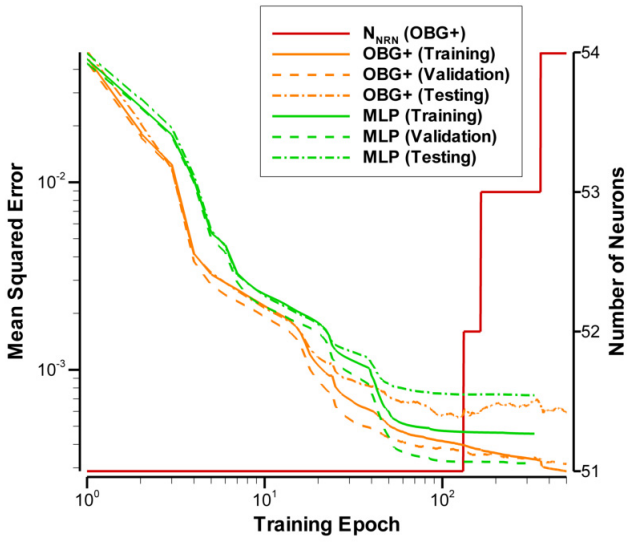


Figure 103: Error variation in transonic modeling (lower surface network)

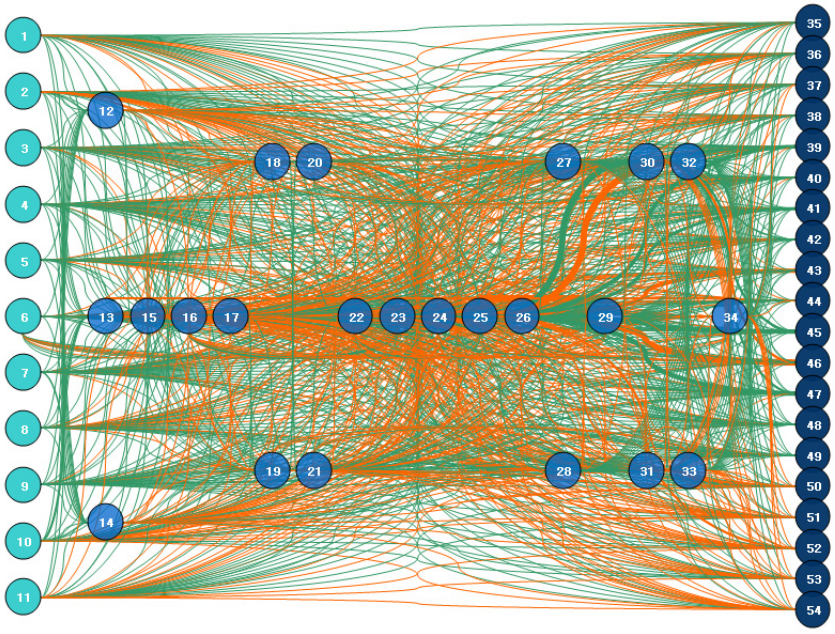


Figure 104: OBG trained network in transonic modeling (lower surface network)

6.5.2 Upper Surface Models

Considering the variation and the nonlinearity in the pressure distribution, the upper surface models in the transonic flow regime requires the most significant training resources when they are compared to the previous cases. Therefore, the number of initial hidden nodes has been varied from 20 to final 50 to determine an appropriate initial network size. For the current modeling, the OBGv+ scheme has been chosen reflecting the observations from the previous chapter showing that this scheme is the most suitable in case of the large number of hidden neurons are allowed initially. Satisfactory prediction performance has been obtained only when using approximately 50 hidden neurons assuming the initial network structure of double-hidden layer MLP. Figure 105, Figure 106, Figure 107, and Figure 108 show the training, validation, and testing error variation both for the OBGv+ training and the BP-LM training with the MLP network. Due to the required computational time significantly increased compared to the previous training cases, the validation error has been monitored and early termination has been executed in case of successive deterioration in the

validation error. Comparing the MLP and the OBGv+ training results, all four cases shown have the similar trend favoring the OBGv+ scheme over the conventional BP-LM training indicating the larger reduction in all three types of errors.

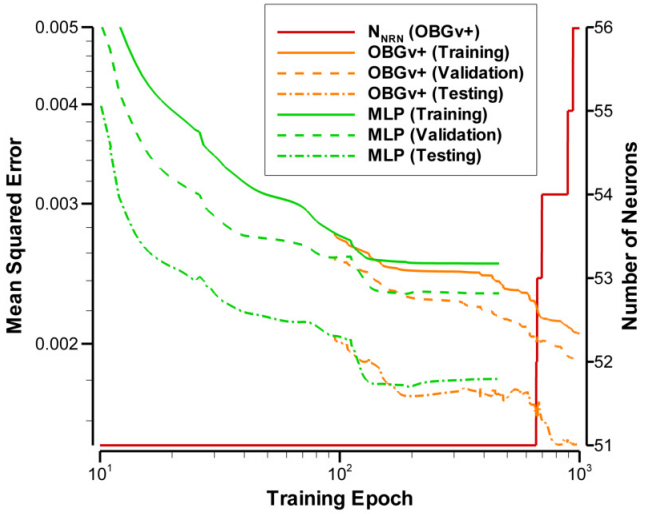


Figure 105: Error variation in transonic modeling starting with 20 hidden neurons (upper surface network)

Figure 109 shows the final network configuration resulted from the OBGv+ training started from double-hidden layer MLP network having 25 hidden neurons in each layer.

More detailed discussion of the training results is given using the training result of the initial 50-hidden neuron cases. Observing individual result for each airfoil reveals the qualitative difference between the two training results by the conventional BP-LM training on the MLP network and the OBGv+ training one. Figure 110 shows the good agreement in the predicted pressure distribution and the target CFD result by both the MLP and the OBGv+ trained networks. This case has no significant difference from the previously modeled subsonic cases due to the absence of the shock involving phenomena. But the modeling capability of networks significantly weakened in the presence of the shock wave. Another 10 cases shown in Figure 111 119

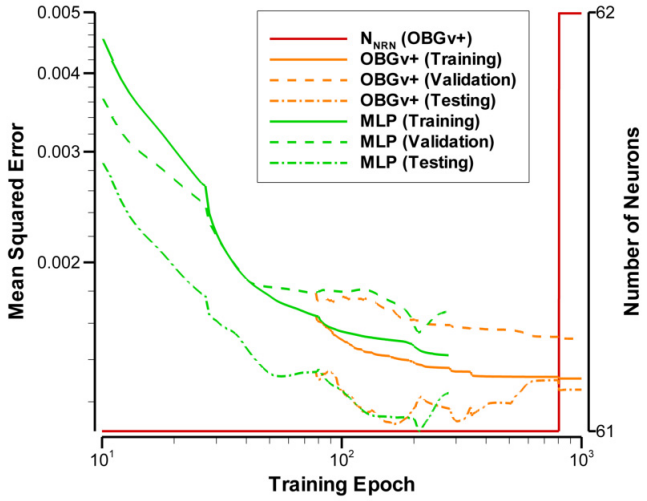


Figure 106: Error variation in transonic modeling starting with 30 hidden neurons (upper surface network)

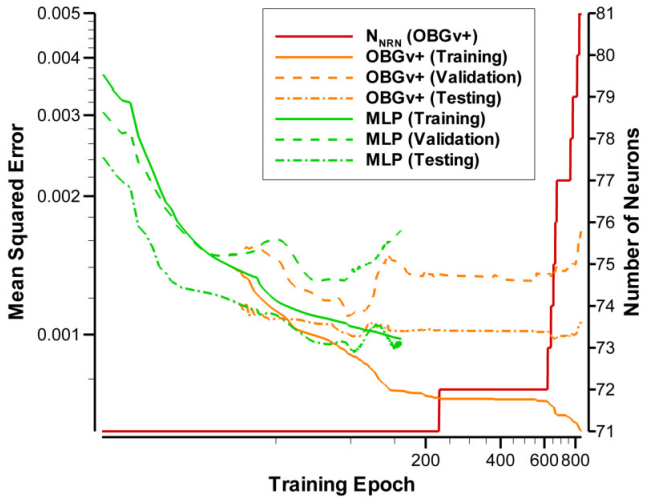


Figure 107: Error variation in transonic modeling starting with 40 hidden neurons (upper surface network)

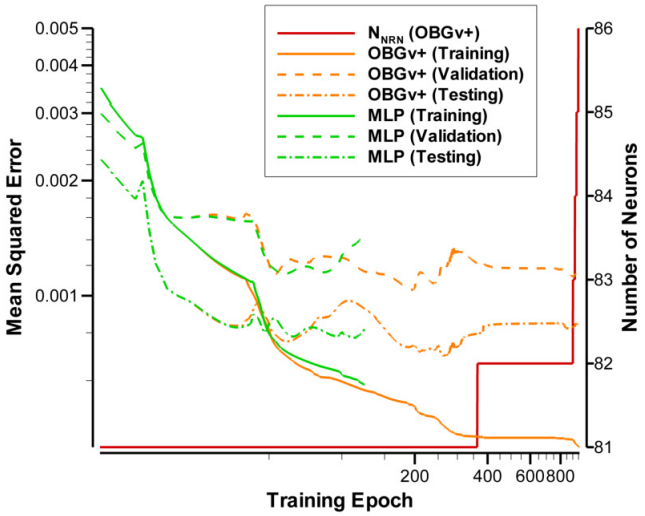


Figure 108: Error variation in transonic modeling starting with 50 hidden neurons (upper surface network)

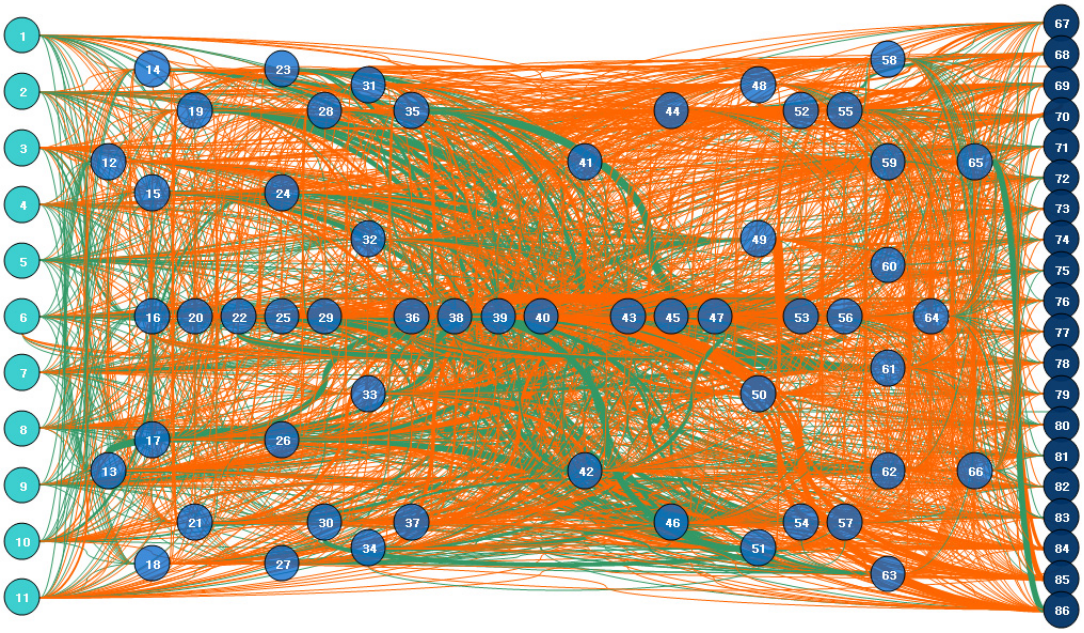


Figure 109: OBGv+ trained network in transonic modeling (upper surface network)

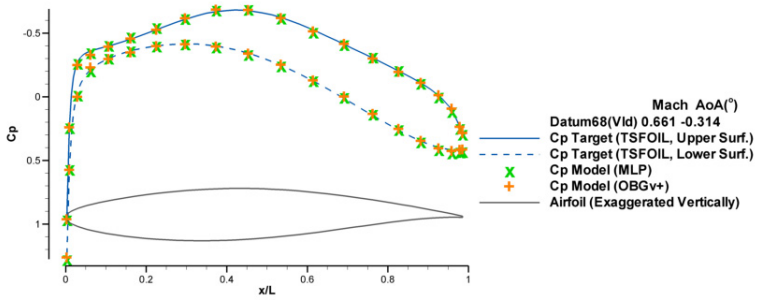


Figure 110: Pressure distribution comparison 1

correspond to those cases. The prediction error of the both models of the MLP and OBGv+ trained network increases compared to the lower Mach number cases, the OBGv+ trained network resulted in the closer agreement with the target pressure distribution near the shock wave.

Figure 125 134 provide randomly selected examples of the pressure distribution comparison with the target data for the MLP and the OBGv+ trained networks combined with the result of the lower surface models. Here, the orange symbols indicate the prediction from the OBGv+ trained networks and the green symbols indicate the prediction from the MLP networks. All examples from training set (Figure 125 130), validation set (Figure 131,132), and testing set (Figure 133,134) show the reasonable agreement to the target pressure distribution in case of absence of the shock wave and the improved modeling capability near the shock wave by the OBGv+ training compared to the conventional BP-LM training using the MLP networks.

6.5.3 Discussion on the Result

The most challenging modeling task was building the upper surface network which has the complicated variation in the pressure distribution due to existence of shock waves. To obtain satisfactory modeling performance in the context of the modeling quality, especially in prediction of accurate position and strength of the shock waves,

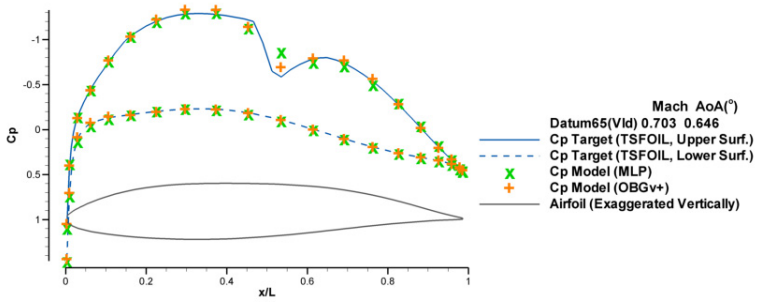


Figure 111: Pressure distribution comparison 2 (in the presence of shock wave)

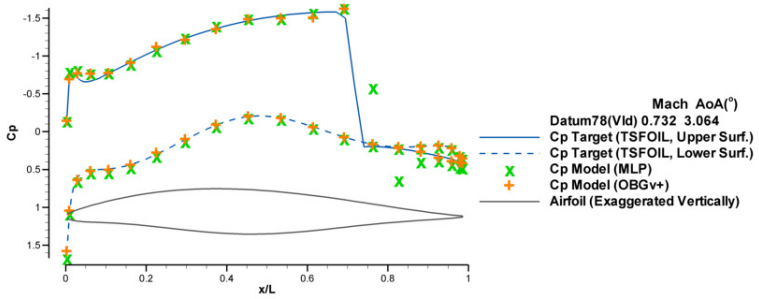


Figure 112: Pressure distribution comparison 3 (in the presence of shock wave)

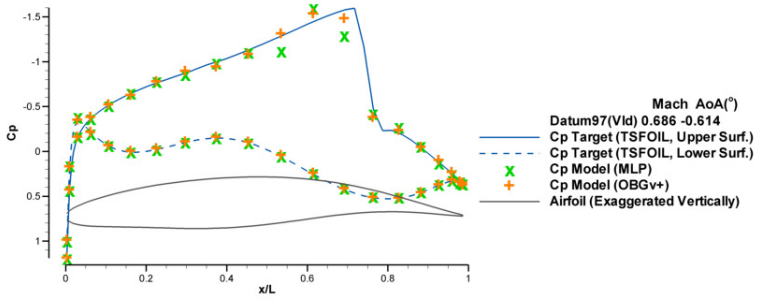


Figure 113: Pressure distribution comparison 4 (in the presence of shock wave)

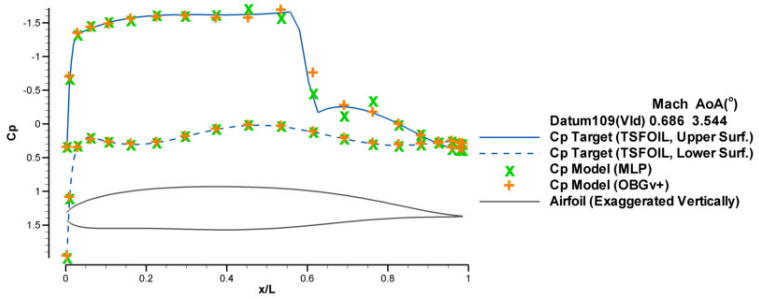


Figure 114: Pressure distribution comparison 5 (in the presence of shock wave)

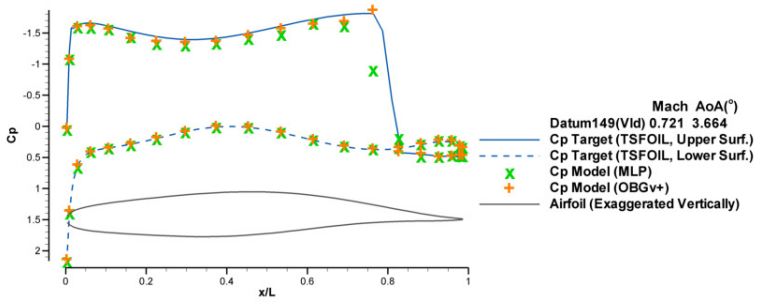


Figure 115: Pressure distribution comparison 6 (in the presence of shock wave)

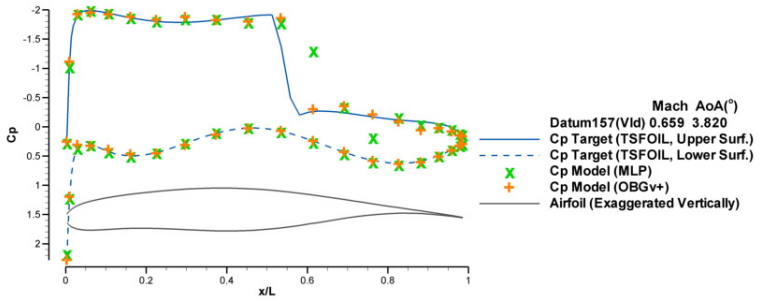


Figure 116: Pressure distribution comparison 7 (in the presence of shock wave)

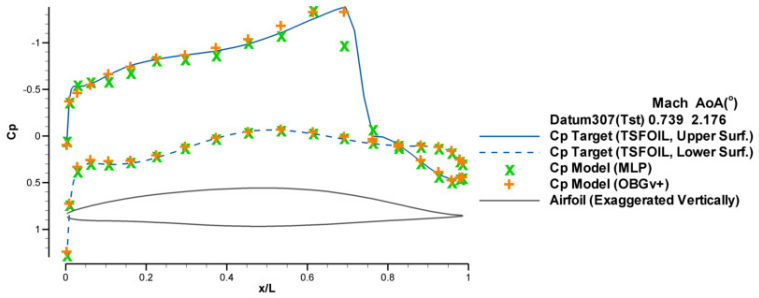


Figure 117: Pressure distribution comparison 8 (in the presence of shock wave)

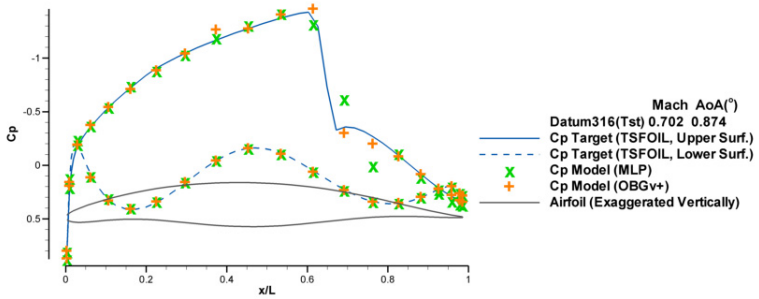


Figure 118: Pressure distribution comparison 9 (in the presence of shock wave)

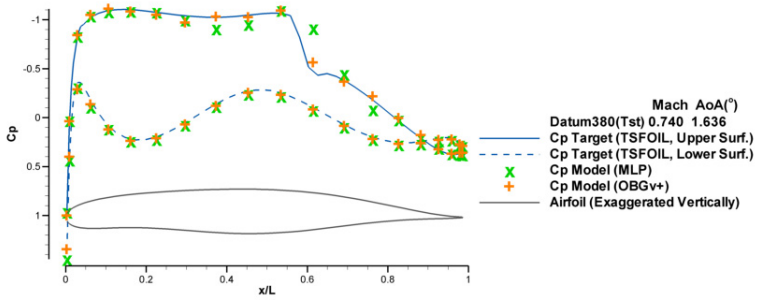


Figure 119: Pressure distribution comparison 10 (in the presence of shock wave)

at least 50 hidden units was required in case of the MLP network. In case of the OBG training, the initial 50 hidden neurons and 1400 connections grew to 56 hidden neurons and 1970 connections and resulted in the significantly increased prediction performance. Figure 120 and 121 show the minimum training errors and the minimum validation errors for all 8 training cases for the upper surface network.

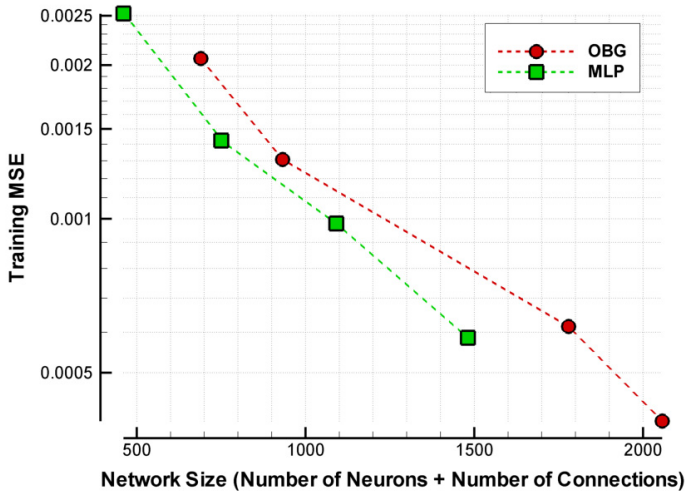


Figure 120: Training performance comparison

The comparison of the prediction performance between the conventional MLP training and the OBG training both started from the identical initial network is provided in Table 12. Using the two networks obtained from the MLP training and the OBG training, first, the 1629 training examples were used to estimate the

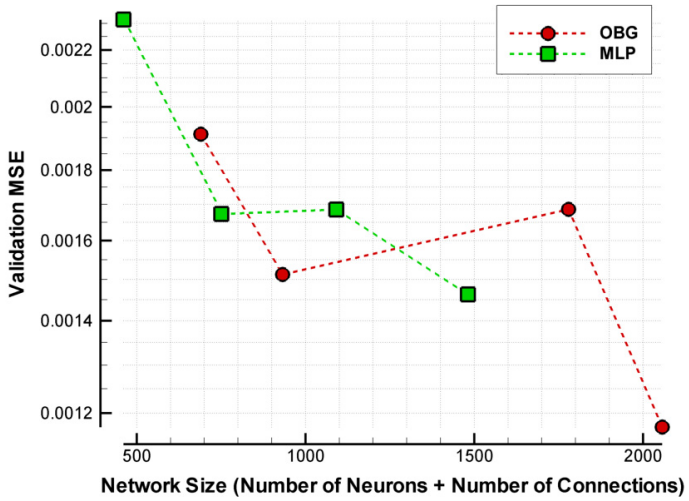


Figure 121: Generalization performance comparison

Table 12: Errors in aerodynamic coefficients (Model versus Target)

	ΔLift Coeff.			ΔDrag Coeff.		
	MLP	OBG	Diff	MLP	OBG	Diff
Training	0.013921	0.010335	3.6 Counts	0.001303	0.000991	4 Counts
Validation	0.014741	0.012690	2.1 Counts	0.001444	0.001229	2.2 Counts
Testing	0.014949	0.014364	0.6 Counts	0.001465	0.001029	4.4 Counts
Total	0.014095	0.010888	3.2 Counts	0.001332	0.001029	3.1 counts

pressure distribution over the entire airfoil for each case (using one common lower surface network). Next, these pressure distributions have been post-processed to calculate lift coefficients and (inviscid) drag coefficients. Each and every case has also been processed to calculate the difference in both coefficients from the corresponding coefficients calculated from the target pressure distribution. The difference values (ΔLift Coeff. and ΔDrag coeff.) in Table 12 are averaged values of those coefficients over the corresponding three data sets; training, validation and testing sets. Here the single *'count'* for the drag coefficient is 0.0001 following the general convention used in aerodynamics literature [39]. The single *'count'* for the lift coefficient is 0.001 [69].

These differences in the lift coefficients between two training methods correspond to 23% of the MLP training error. Also the drag coefficients calculated from the

pressure distribution model of the OBG method has 23% more accurate in average.

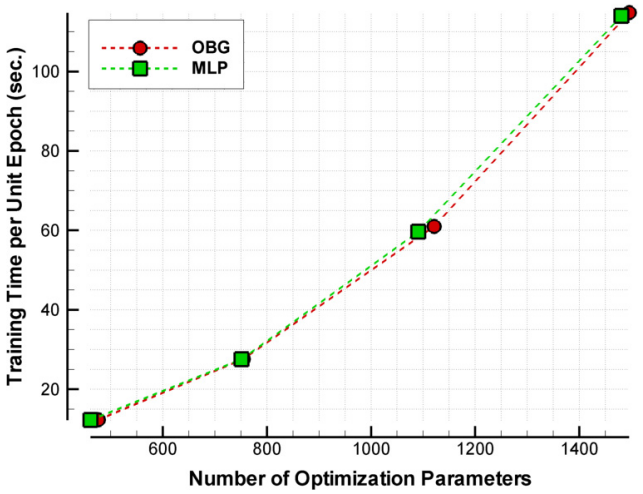


Figure 122: Unit training time comparison

Although the required absolute training time is longer in the OBG training than the MLP training, the network sizing capability of the OBG method can significantly reduce the need for multiple trainings which is usually required in case of the MLP training. This advantage has already been observed and described in the fifth experiment. Moreover, by maintaining the actual number of optimization parameter as same as the initial network's, the unit computation time for each training epoch is virtually identical to the initial MLP network's. Figure 122 proves this showing that the unit training time for OBG cases which have hundreds more network connections than the corresponding MLP cases can advance each epoch without relative increment in computation time. The increase in prediction run time of the OBG networks is inevitable due to the grown network size compared to the baseline MLP networks. Figure 123 shows the impact of the overall network size to the single run time of the models for a prediction step.

Figure 124 also shows the impact of the number of neurons to the same quantity. Two figures show that the model run time (or prediction time) is a strong function of

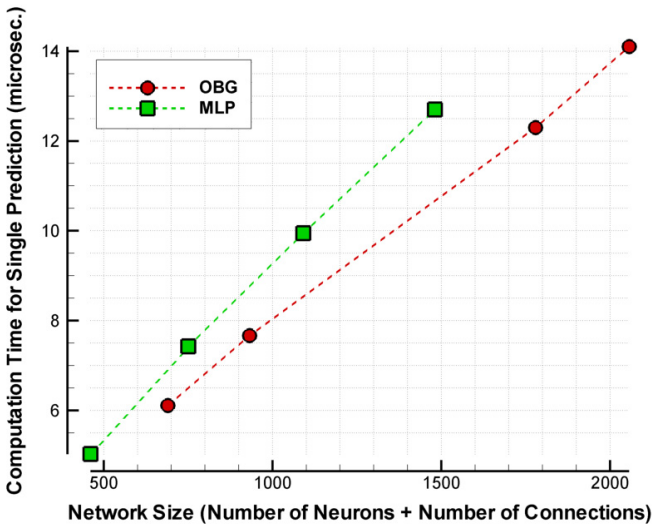


Figure 123: Model prediction time comparison 1

number of the neuron in the network and only loosely related to the number of the connections.

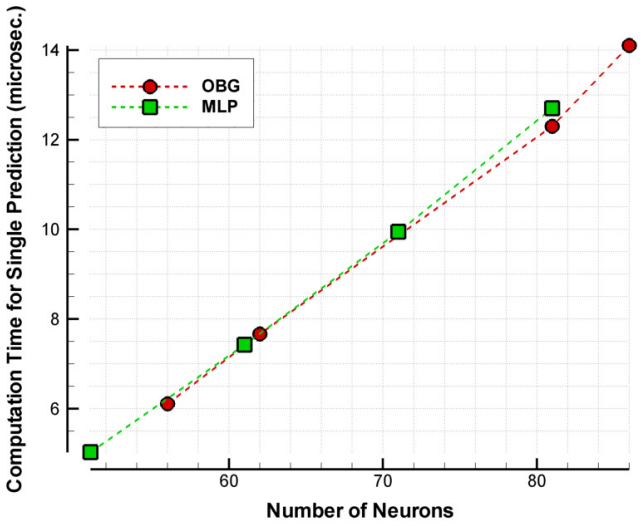


Figure 124: Model prediction time comparison 2

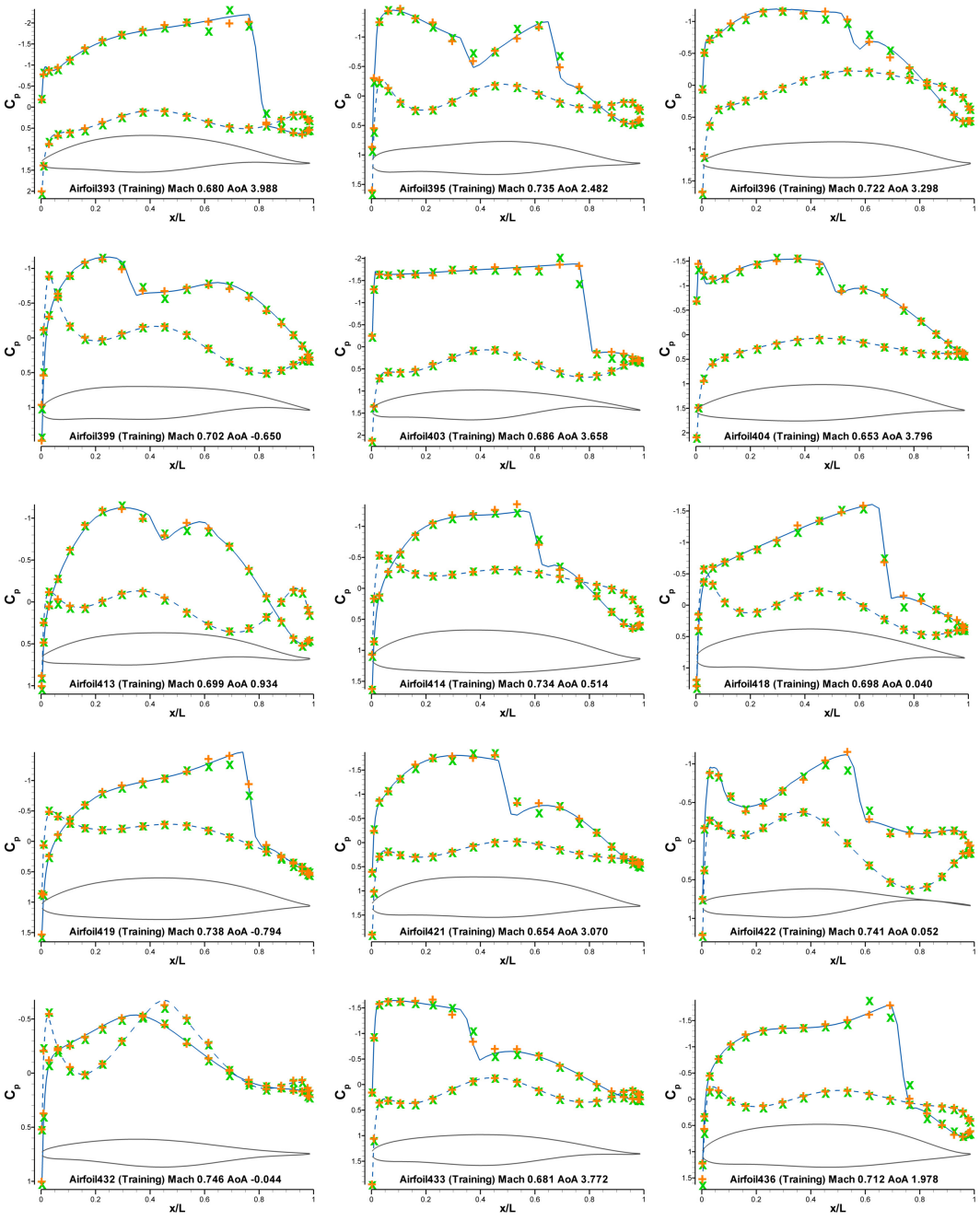


Figure 125: Transonic airfoil modeling result (training samples 1)

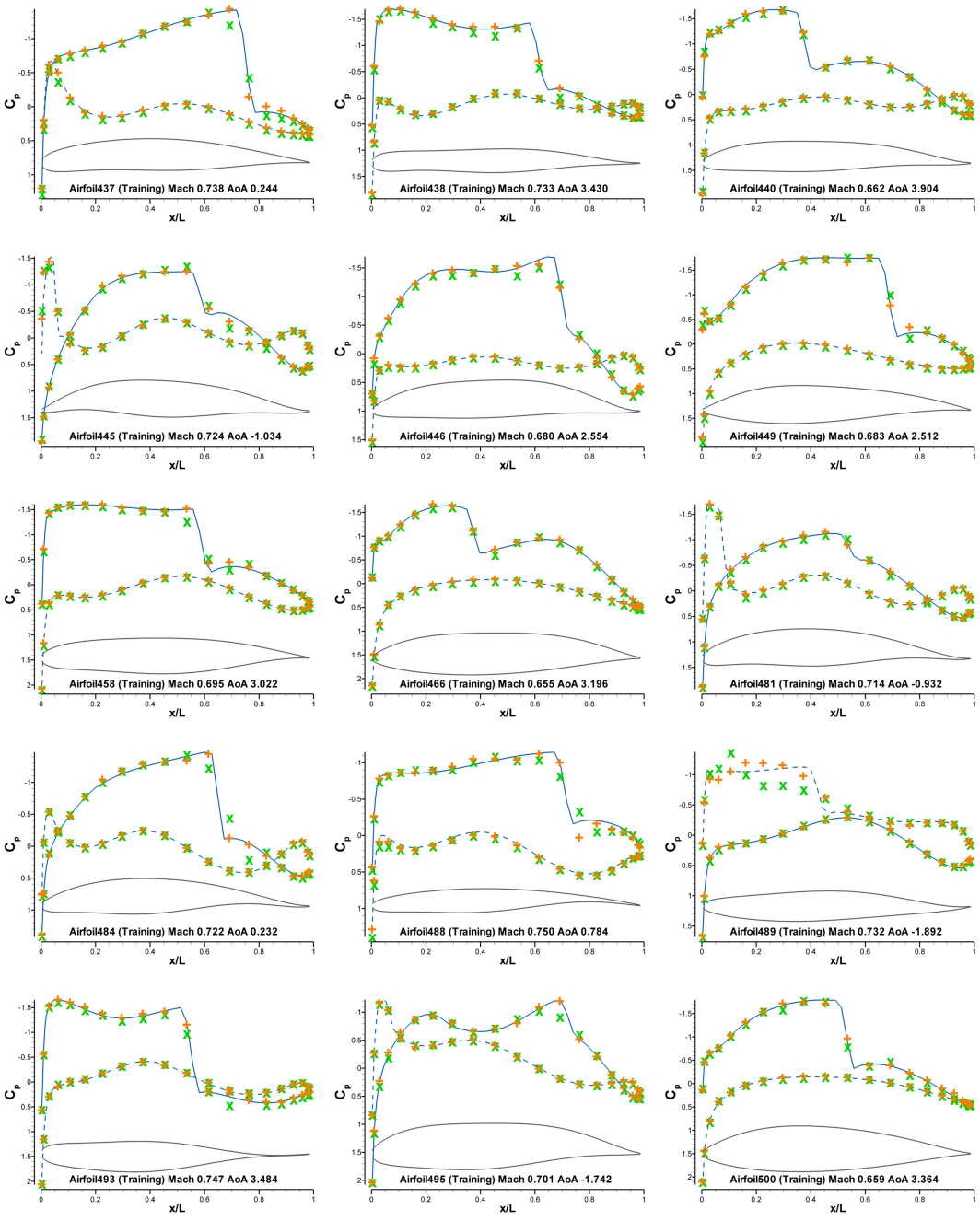


Figure 126: Transonic airfoil modeling result (training samples 2)

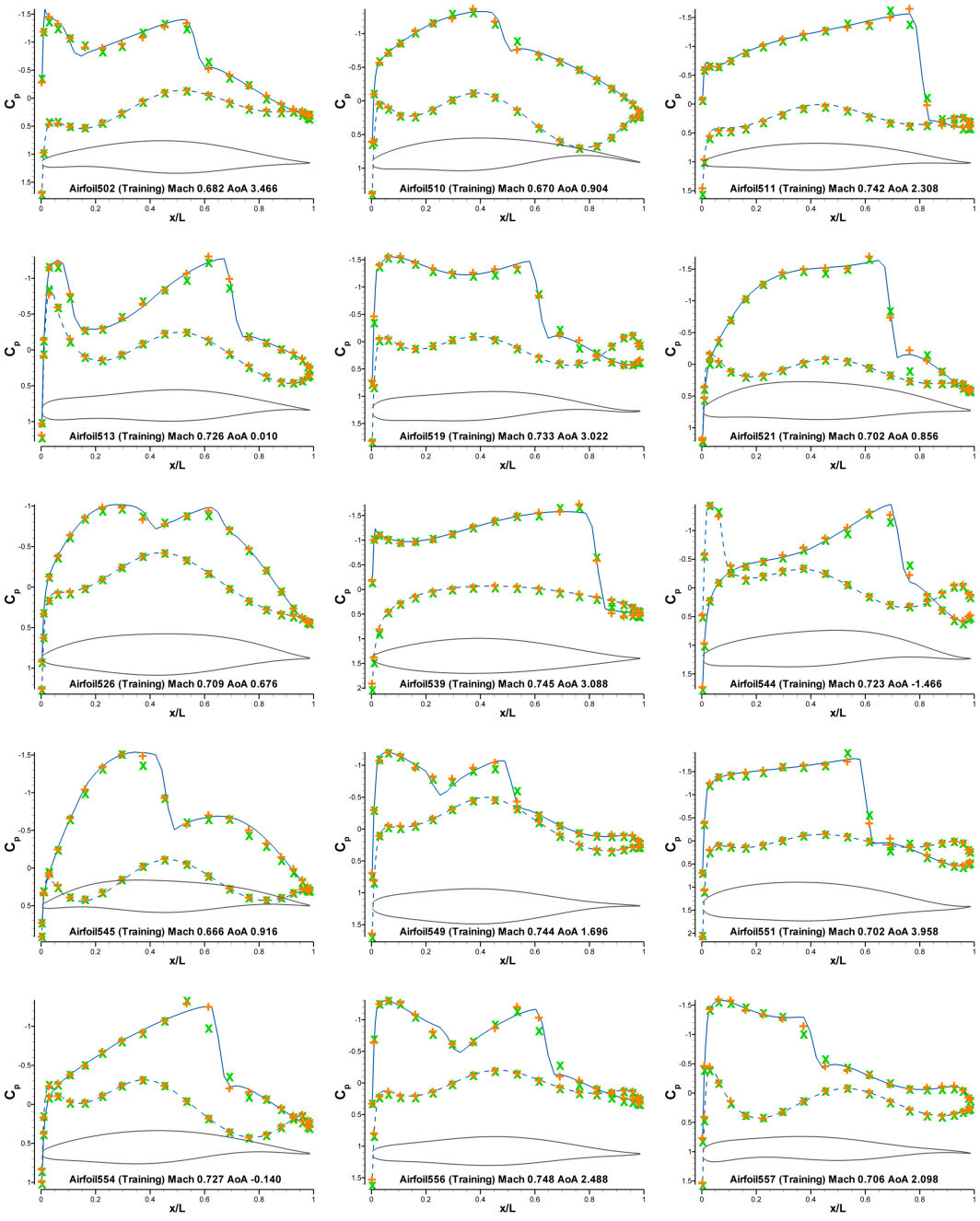


Figure 127: Transonic airfoil modeling result (training samples 3)

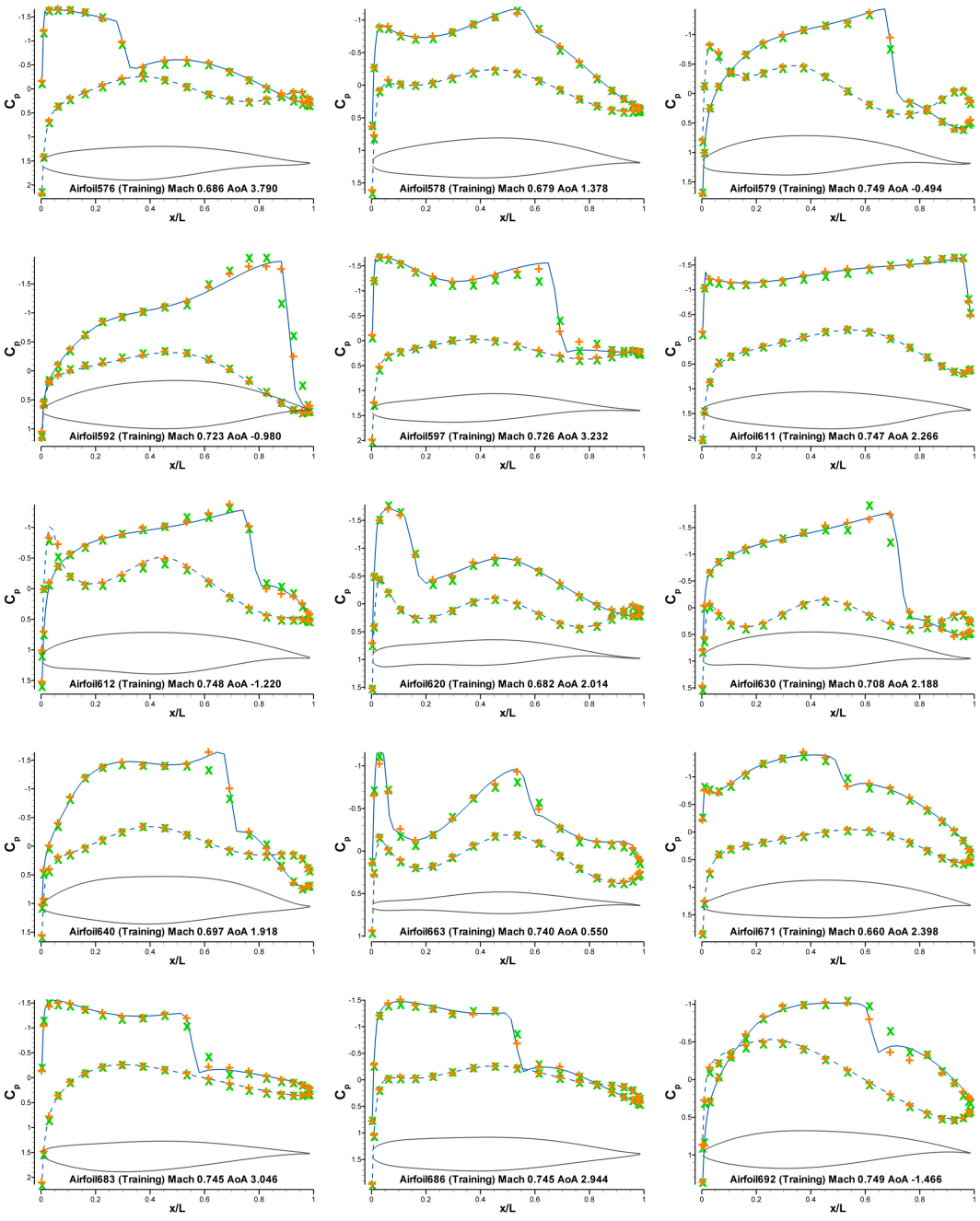


Figure 128: Transonic airfoil modeling result (training samples 4)

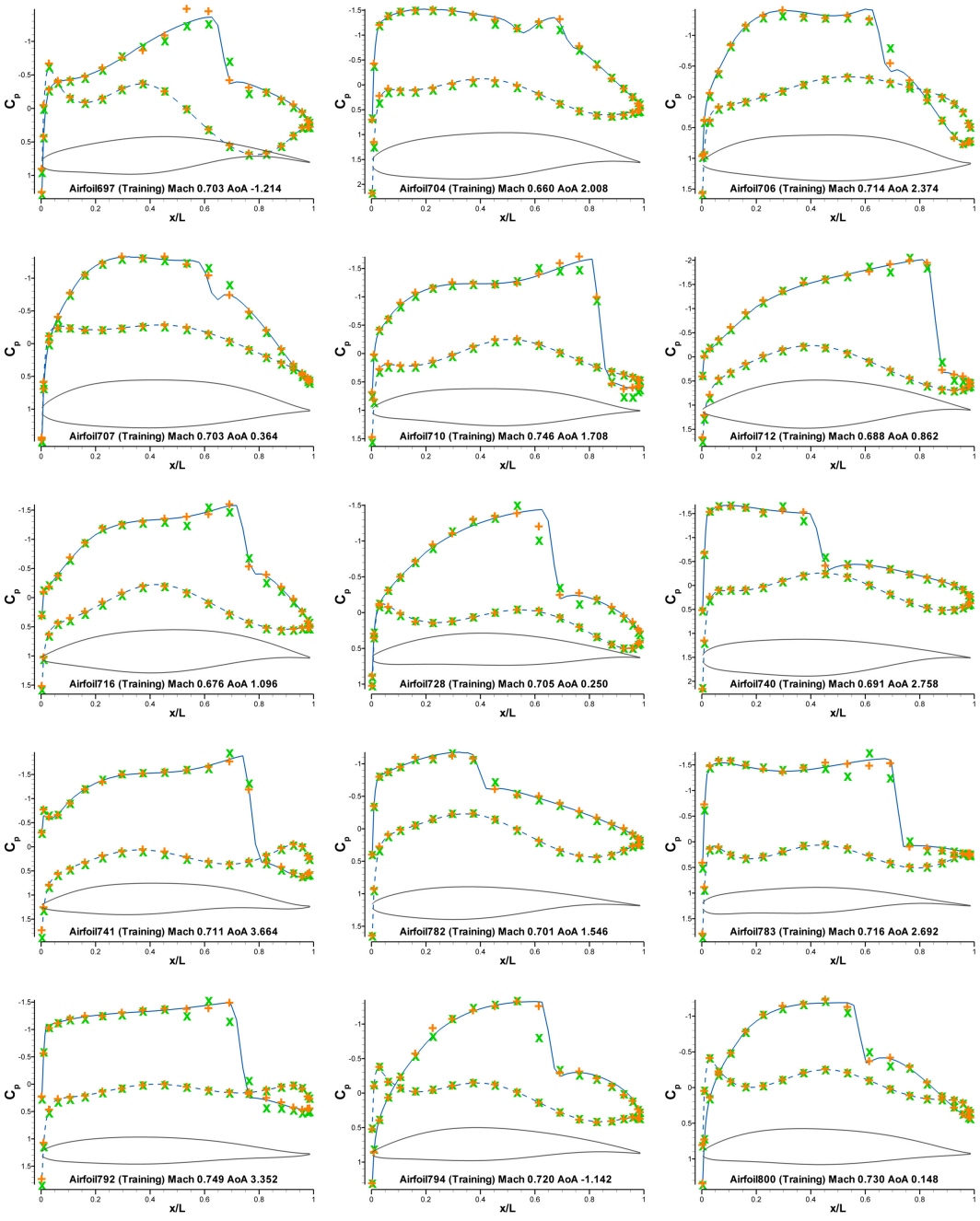


Figure 129: Transonic airfoil modeling result (training samples 5)

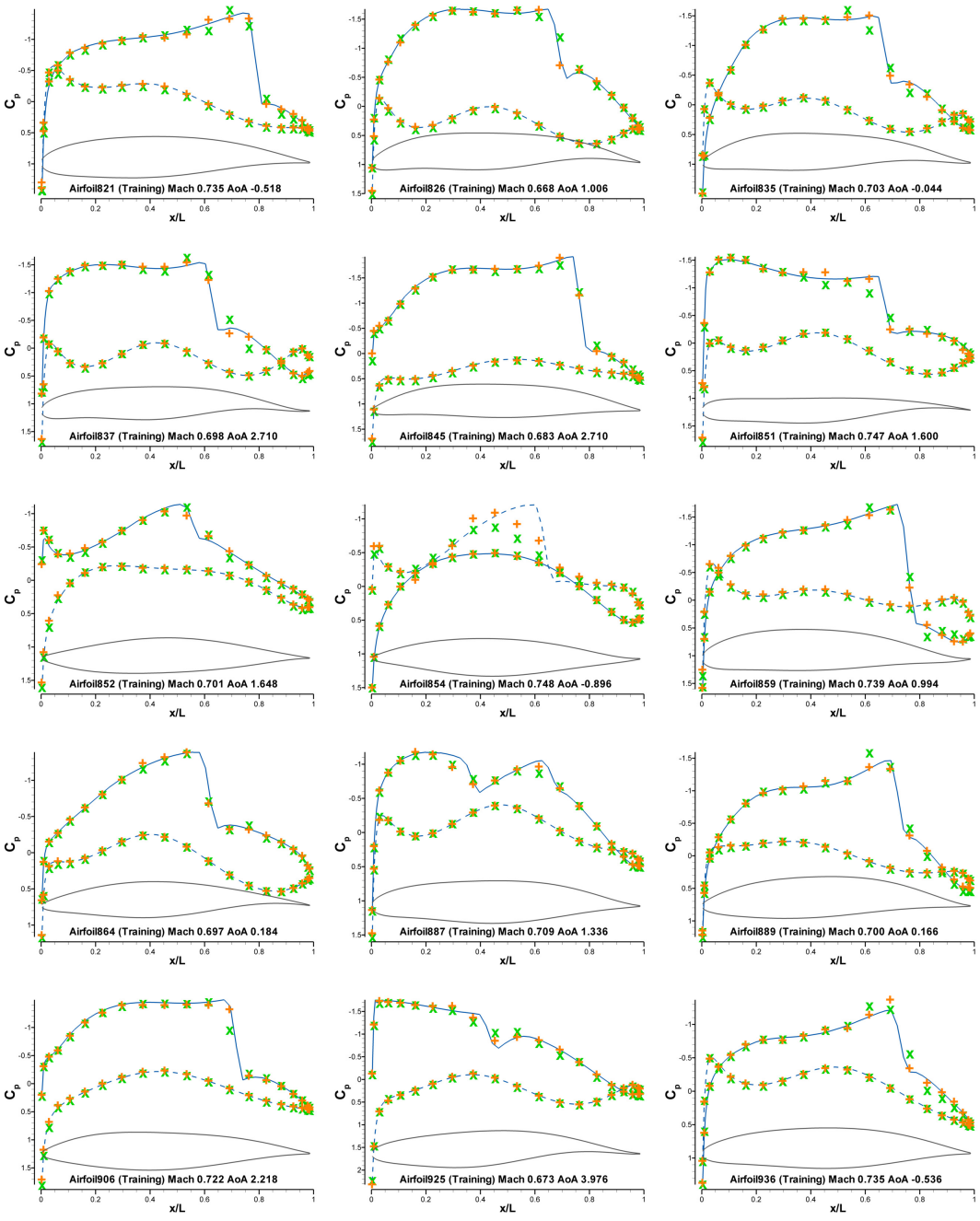


Figure 130: Transonic airfoil modeling result (training samples 6)

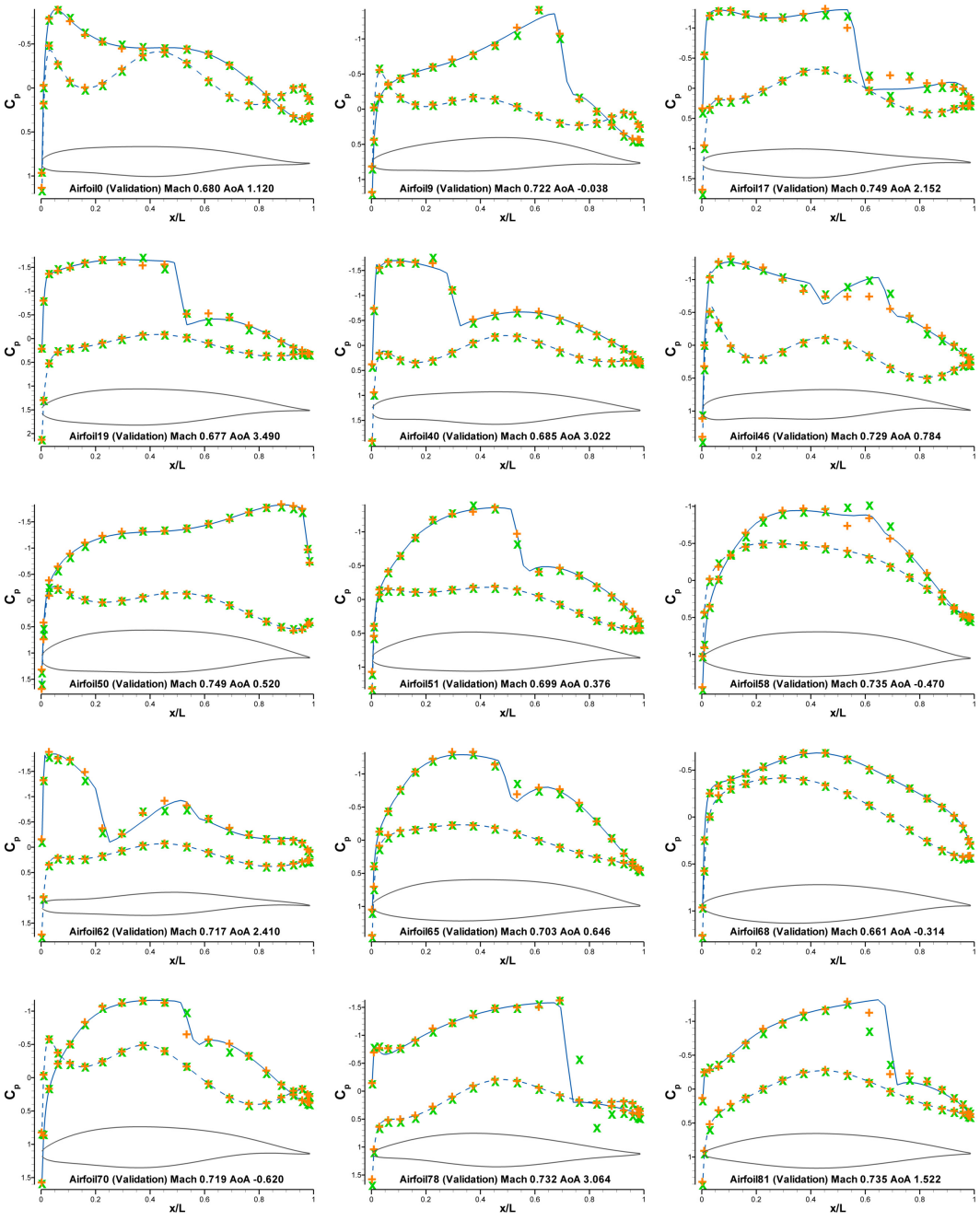


Figure 131: Transonic airfoil modeling result (validation samples 1)

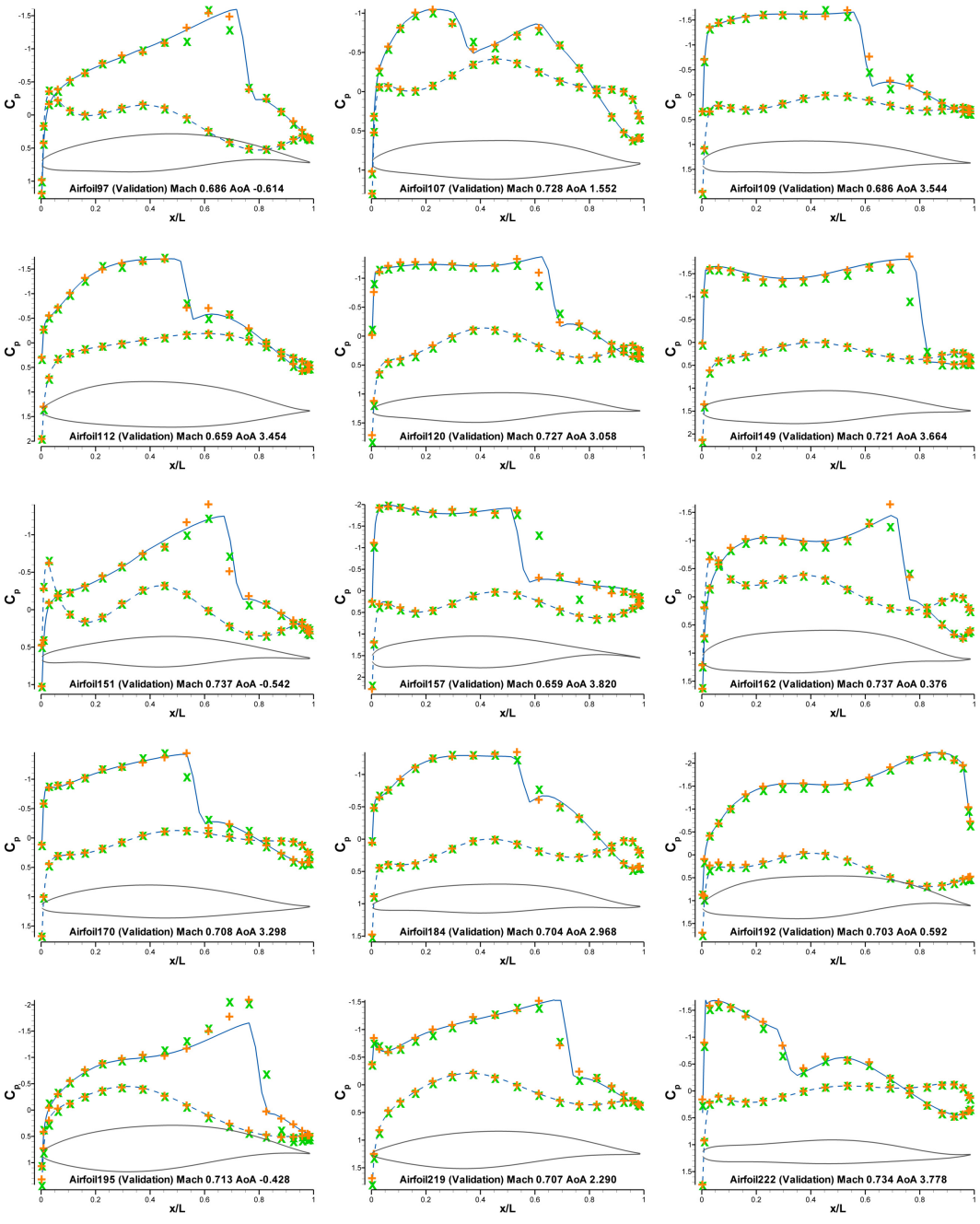


Figure 132: Transonic airfoil modeling result (validation samples 2)

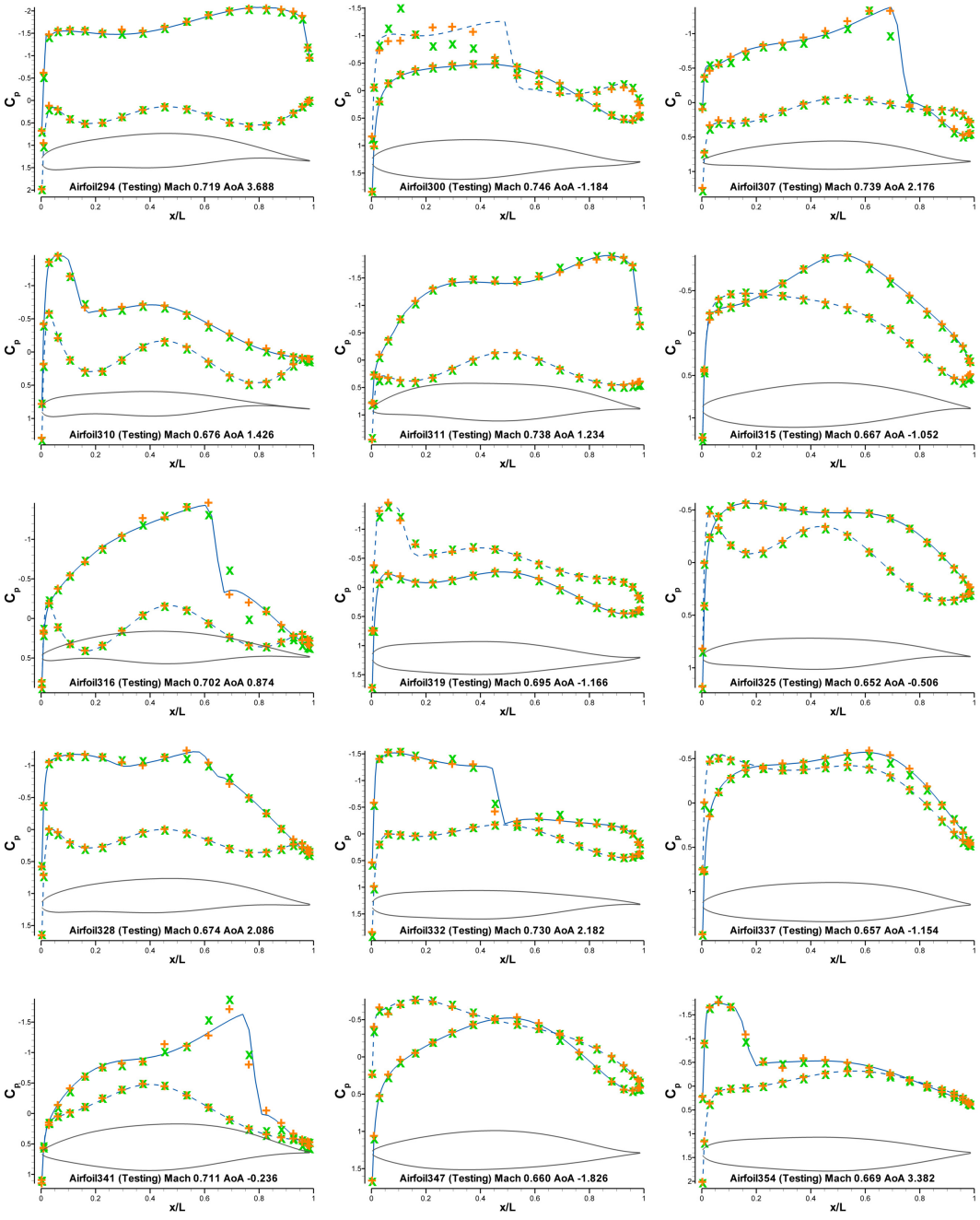


Figure 133: Transonic airfoil modeling result (testing samples 1)

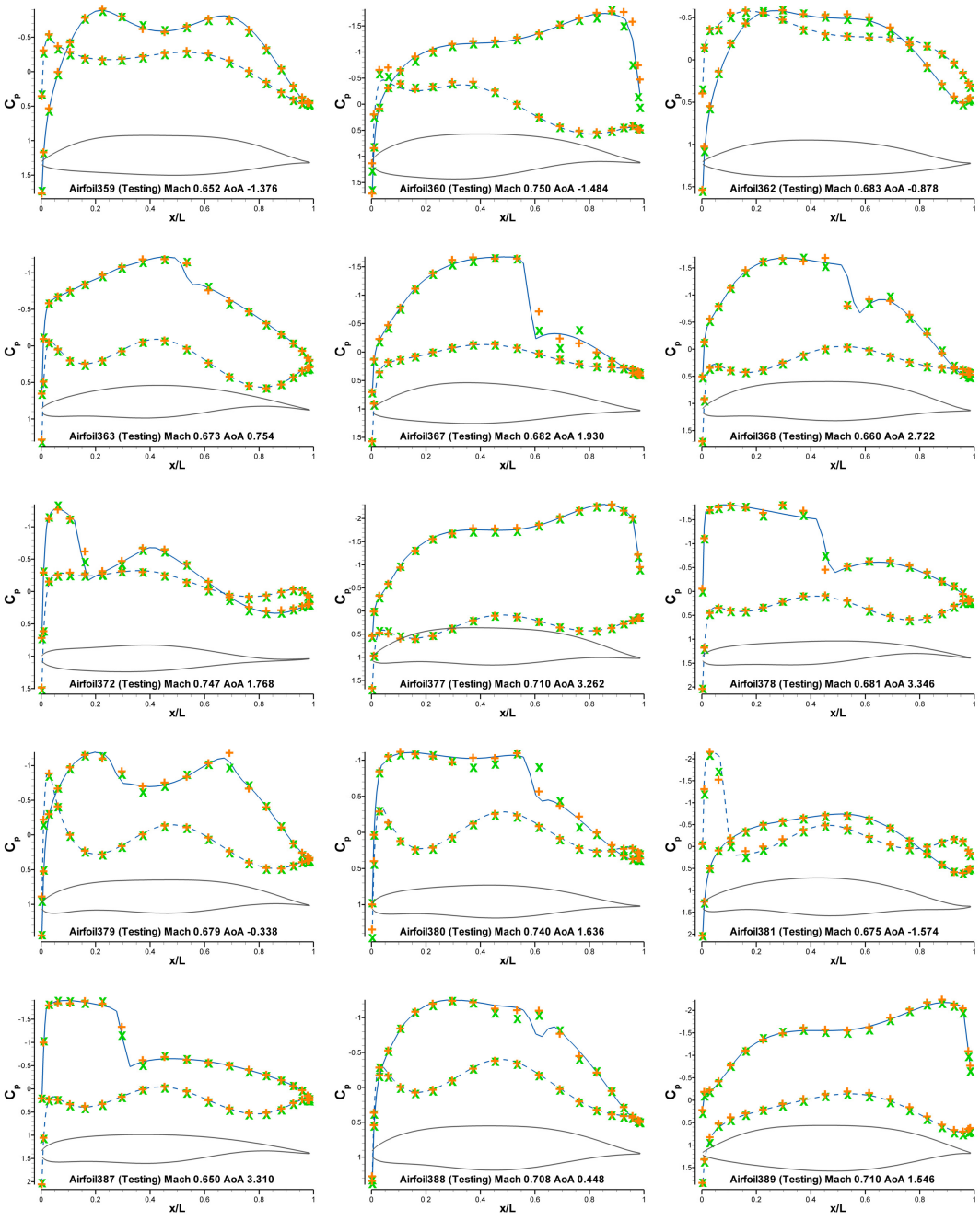


Figure 134: Transonic airfoil modeling result (testing samples 2)

CHAPTER VII

CONCLUSION

Considering the fundamental multidisciplinary and intensive data-driven nature of the aerospace research and development procedures, the importance of the mathematical modeling is paramount. The surrogate modeling using artificial neural network is widely applied for the variety of aerospace modeling tasks, but there still is great potential for the further improvement in its capability and efficiency. The current thesis has been motivated by questioning the conventional paradigm of neural network training procedure which is mainly focused on the optimization of the connection weights assuming the predefined network structure based on the belief that the optimization of the connectivity has also great leverage on the generalization capability of the neural networks. This belief has been inspired and supported by mathematical, biological, and computational observations. Existing network design methods have been reviewed and have led to find the common limitation in view of the weight-connectivity dichotomy constraining network design efficiency by decoupled optimization or learning hierarchies. The concept of *connectivity adjusting learning scheme* has been established contrast to the conventional weight-only adjusting learning rules and investigation on the relation between the establishment of the connectivity and the weight variation has resulted in a particular learning algorithm named as the *Optimal Brain Growth* which adjusts connectivity as well as weights towards local minimum of the training error. This novel algorithm has been tested on multiple canonical regression and classification problems indicating promising modeling capability and efficiency compared to the conventional BP-LM training using the MLP networks. To augment generalization capability of the OBG

algorithm, validation information has been exploited resulting in the OBGv+ scheme which consistently outperformed single- and double-hidden layer MLPs during the series of comparative studies. As a more practical surrogate modeling task for the physics-based analysis, the geometry-to-pressure mapping of the transonic airfoil has been performed using the conventional BP-LM training and the OBG training. A significantly improved modeling capability has been resulted in case of the OBG trained surrogate models for the challenging problem of appropriate detection and modeling of the shock wave.

7.1 Recapitulation of the Thesis

In Chapter I, the important role of the surrogate modeling as a key enabler to extend the limitation of the multidisciplinary design optimization methodology has been reviewed and among the various surrogate modeling techniques, particular attention to the artificial neural network has been given by describing its distinctive characteristics and advantages. The basic mechanism and possible evolution of the neural network has been reviewed and the challenging issues of network design and scalability problem of the conventional neural network training method have been introduced.

In Chapter II, contemporary practice of surrogate modeling for computational fluid dynamics (CFD) analysis has been reviewed revealing the inconvenient problems of settling with the particular network architectures limiting the modeling performance by that settlement. The mathematical theories on the primary factors affecting the generalization capability of the neural networks, the practical limitation of the *universal approximator* theorems, and the dissimilar performances resulted from the different connection architecture support the significant impact of the network topology. After reviewing fundamental two algorithms, error back-propagation algorithm and the Levenberg-Marquardt algorithm, which are indispensible for the

conventional Multilayer Perceptron (MLP) network training and for its further improvement, three main groups of methods which are available today, neuro-evolution, constructive and pruning methods, to explore topological space have been reviewed.

In Chapter III, an alternative to the current topological exploration methods has been pursued by observing the source of numerical inefficiency of the current methods. Major finding is that, as long as the connectivity and weights are dealt with by separated means, potentially more efficient topological exploration is hard to be achieved. In this way, the need for the *connectivity adjusting learning scheme* has been justified. Interestingly, in the biological field of research, the *activity-dependent mechanism* has been known to actively participate in the reformation of the connectivity in the biological neural network as well as the synaptic strengthening.

By interpreting an instance of the absence of the connection as a zero-weight connection, a meaningful mechanism of connectivity growth has been conceptualized. Moreover, just slight extension to the classical back-propagation algorithm has been found to enable the selection of the new *optimal* connectivity by allowing the instant measurement for the derivative of error with respect to the weight of each and every feed-forward connection in the network regardless of whether it is a currently established connection or not.

Combined with the fixation strategy for the less-error-reducing connection weights preserving the total number of the adjustable parameters, a unique connectivity adjusting learning mechanism, *Optimal Brain Growth*, has been conceptualized which is compatible with the efficient second-order weight optimizer.

Based on the possible seamless integration of the previously separated two learning mechanisms for connection weights and connection structure, growing strategy starting from a possible minimal network structure has been hypothesized with an additional mechanism for adding new computational units with minimal connections, based on the correlation maximization or the neuronal decomposition.

In Chapter IV, the conceptual OBG learning mechanism has been translated into the mathematical formulation based on the existing derivation of the error back-propagation algorithm. Particular rules for connectivity adjustment, additional treatment for the compatibility with the Levenberg-Marquardt algorithm, and detailed algorithm for neuronal addition to the network have been given.

In Chapter V, first, a simple fluid mechanics problem for pipe flow modeling has been tried by the conventional training algorithm using the MLP network and the previously formulated OBG training method. Second, the classical two-spiral classification problem has been tried to be solved by the similar comparative way. In both experiments, the OBG has outperformed the MLP networks in its chance to the viable solutions and in the training efficiency. Using the Complicated Interaction Regression Function, the generalization capability of the OBG and the MLP trainings have been compared leading to the observation that the advantage of the OBG training is more evident in the cases of trainings started with relatively small networks. To augment generalization capability in case of the larger networks, adding neurons and controlling the activation of the OBG scheme from the conventional BP-LM training using the validation information have resulted in the variation of the OBG scheme, the OBGv+ scheme, and this training algorithm has consistently outperformed the conventional BP-LM training algorithm in the measurement of training, validation, and independent testing error.

In Chapter VI, a more practical surrogate modeling for physics-based analysis has been performed, i.e., geometry-to-pressure mapping of transonic airfoil. The CFD analysis using TSFOIL code has produced the thousands of training examples mapping airfoil geometry and flow conditional variables to the distributional information of the pressure coefficients. As a geometric representation method for the transonic airfoils, PARSEC method has been adopted which defines an airfoil using 9 geometric parameters. For each surrogate modeling case, two separated networks for upper and

lower surface have been trained to overcome the limitation of the available computational resources. Both for the subsonic and the transonic flow regimes, the OBG training resulted in the more accurate surrogate models, and the distinction between the MLP and the OBG training is clearer in the presence of the shock wave.

7.2 Summary on Research Questions and Hypotheses

7.2.1 How to Explore Topological Space?

Thorough literature review on the way contemporary methods explore the topological space showed two distinctive groups of methods. One is the network sizing methods including the constructive methods and the pruning methods. These methods need total re-training processes after introductions of new network components or operate within a basically identical, particular network topology. The other group of methods is the evolutionary approach to evolve task-specific networks exploiting a population-based heuristic search. The common practice in both sets of methods is based on *weight-connectivity dichotomy*. Adopting two hierarchically dissimilar search methods for the weight optimization and for the topological search, applicability of those methods to practical network design tasks is quite limited such as to small-sized networks.

7.2.1.1 Connectivity Adjusting Learning Scheme

The developed OBG method is an instance of the connectivity adjusting learning scheme which learns optimal internal connectivity as well as optimal weight set in the seamless and integrated way. The experiments have shown that OBG can bring an improved learning capability without significant increase in computational cost for many cases including multiple regression problems and one classification problem. Contrast to the evolutionary approach which requires computationally expensive multiple evolutionary loops, the OBG method completes the weight optimization and the

structural improvement in terms of training capability simultaneously, within a single training campaign. The OBG method also has competence to the conventional network sizing methods by enabling more genuine topological search beyond the network sizing functionality. Therefore, the connectivity adjusting learning scheme has potential to be very efficient approach for the topological search of the vast network design space.

7.2.2 How are Weights and Connectivity Linked Together?

In the current paradigm of neural network training, weights and connectivity are usually dealt with as two independent entities. But the attempted development for a connectivity adjusting learning scheme requires an investigation to find useful mathematical model on the relationship between the discrete connectivity and the continuous weights.

7.2.2.1 Concept of Latent Connections

The concept of latent connection has been conceptually established and algorithmically formulated in this thesis. The physical definition of the *latent connection* is any feed-forward connection which has currently zero-weight value in a network. When a latent connection is considered as a candidate new connection, the relationship between the connectivity and the weights can be established. Extending the standard BP algorithm, the derivative of training error with respect to each and every weight can be obtained regardless of its validity as a present weight. Utilizing this information, it is possible to make a rational choice between an adjustment for a present connection and a creation of a new connection. In this way, the continuous change of weights and the discrete creation or pruning of a new connectivity are mutually interrelated and affect each other. The operation of the OBG method and its improved learning capability which have shown in the series of experiments prove that the relationship between the weights and the connectivity based on the concept of

latent connection is not trivial one.

7.2.3 Which Connections to Grow?

The application of the concept of the latent connection to find a better connectivity will result in a monotonical growth in the connectivity of a network. Hence, the efficient criteria to control the growth of a network need to be found.

7.2.3.1 Optimal Brain Growth

The hypothesized growth mechanism adopting a partial weight optimization is the essence of the OBG method. In OBG, the high-gradient latent connections are promoted to be established as a new connection and the less-changing or stabilized connections are promoted to be pruned from a network. This pruning mechanism can be selected between the hard pruning and the soft pruning. The hard pruning physically eliminates one of the existing connections and is only possible when the candidate weight for pruning is near its zero value under the numerical threshold. And the soft pruning means the exclusion of the weight from the set of optimization parameters, which allows the connectivity of a network grow monotonically but maintains the computational cost for the training approximately same as the initial network. The experiments proved that both ways of network pruning can be useful to minimizing the computational cost during the training process.

7.2.4 How to Add a New Neuron?

To exploit the connectivity adjustment capability of the OBG method, two hypothetical mechanisms have been devised.

7.2.4.1 Correlation-Based Decision

The experiments adopting this mechanism showed that the addition of the computational units extends the learning capability of a network resulting in the further reduced minimum training error.

7.2.4.2 Decomposition of Existing Neuron

The experiments adopting this mechanism also showed that the addition of the computational units successfully refines the current network's performance and extends the learning capability of a network resulting in the further reduced minimum training error.

7.2.5 How to Train a Growing Network Efficiently?

This research question can be translated into how to maintain computational efficiency of the hypothesized method as one instance of the general constructive method.

7.2.5.1 Partial Optimization Scheme is Required

Considering the algorithmic characteristics of the (pseudo-) second order optimization methods for the weight optimization process, the OBG method adopts the partial optimization strategy which freezes the stabilized parameters. Through the series of the experiments, this strategy contributed to the computational cost during the training process is almost always maintained to that of the level of the corresponding sizes of the conventional MLP network. Almost identical *unit computation time* (computation time per training epoch) have frequently been obtained through the experiments with the MLP training. In other words, the OBG method utilizes the considerably increased network connections but, by the mechanism of the soft pruning adopting a partial optimization strategy, the computation time for the weight optimization at each training epoch is maintained to that of the initial network.

7.3 Contributions and Recommendations

The *Optimal Brain Growth* algorithm is a valuable addition to the *library* of the contemporary neural network training algorithms as one of a few available algorithms consistently learning optimal connectivity as well as optimal weights possessing similar level of numerical efficiency with the most common network training algorithm

such as BP-LM training method. It enables not only the faster convergence to the locally minimum training error without consuming significantly more computational resources but also higher-level synthesis of networks. The common preference for the strictly-layered structure such as MLP over the more densely connected one such as FCC is, obviously, due to the limitation on the available computation resources, i.e., time and memories required in training and prediction phases. For the given number of the constituent neurons in a network, the FCC corresponds to the maximal number of connections although the FCC configuration turned out as the most compact form which is capable of learning certain types of problems such as N-parity problem [90]. The OBG training typically converges to the intermediate network configuration between the *efficient* MLP and the *powerful* FCC as the result of simultaneous learning in the connectivity as well as the weights. The challenging problem of scalability in the MLP training is more formidable when the additional hidden layer is required, by which the number of network parameter can increase exponentially. In other words, the scalability problem in the conventional MLP training is, at least partially, due to the discreteness in the network design practice, i.e., every new computational unit has to come with full connections to the adjacent layers and every new hidden layer has to come with a group of such units. The greatest potential of the OBG training method is in its ability to navigate the vast network design space in more continuous manner. The extended BP algorithm exploiting the latent connections enables a connection-by-connection growth (in case of soft pruning) or adjustment (in case of hard pruning). The neuronal decomposition with local reinitialization enables a neuron-by-neuron growth without the burden of full-adjacent connections. These two mechanisms operate refining the previously learned network status only further reducing the training error, without the necessity of re-training processes. Although all three benchmarking test problems described in the previous section are relatively simple and small-scale problems, the OBG training results have consistently shown

the improved learning capability without significant increase in computational cost compared to the conventional weight-only trainings using fixed MLP structures. As a local search method strictly dependent on the gradient information, the OBG is not expected to discover surprisingly better network structure and is also bound to the quality of the initial network's as the conventional MLP training. But, in conjunction with the evolutionary approach which currently adopts the conventional weight-only training methods, the OBG can be synergically exploited to promote more efficient evolution by providing the improved learning capability. Following research issues are recommended for the further investigation of the developed algorithm;

- Modification of the OBG algorithm to the *on-line* learning applications extended from the current *batch* learning algorithm; The OBG can be extended to on-line learning as a general learning rule straightforwardly and, depending on the size of the training set and the nature of the problem, this approach might be more suitable than the batch learning.

- Combination with the pruning method; In case of combining an efficient pruning algorithm, the OBG can be enhanced in its generalization performance and prediction speed. For this extension, penalty term type algorithm is more suitable than the sensitivity-based ones to eliminate the need for re-training process. But, in case of using the LM algorithm as a weight optimizer, an efficient numerical algorithm has to be devised to be infuse penalty term into the form of the squared-error cost function.

- Exploitation of the OBG algorithm in the context of the neuro-evolutionary approach; evolution and learning are two most fundamental forms of adaptation which is mutually beneficial [62, 12, 86, 89]. The OBG algorithm is expected to significantly reduce the required scale of the evolutionary exploration by facilitating enhanced local search not only in the weights but also in the topological

aspect.

- Ensembling of multiple networks; Contrast to the currently common way of network ensembling which simply merge the pre-built networks [94, 82], a novel ensembling method might be possible not only merging the pre-built networks but also enabling synthesis of the connectivity using the OBG algorithm.

- Multi-stage training; Contrast to the conventional learning algorithms which memorize the modeling data only in the way of connection weight values within the limitation of the given structure, the OBG can memorize the same modeling data not only in the weights but also in the more permanent connection structure. In case of performing multiple stages of training campaigns such as from the simple problems to the relatively more complicated problems, the OBG algorithm can facilitate incrementally complexified connection structure.

7.4 Oil, Vinegar, and Vinaigrette

Cline surveyed the experimental evidences which support the role of the activity-dependent mechanism on the reformation of the network connectivity in the biological organisms and concluded as follows [23, 16]; *"There is a tendency to believe now that Sperry and Hebb is not oil and vinegar, but maybe vinaigrette."* Here, Roger Sperry represents the invariable destiny of the connections between particular two parts of the nerve systems and Donald Hebb is the originator of the Hebbian learning rule which emphasizes the variability in the synaptic strength depending on the correlation in their activities of two parts of the nerve system. In the context of the artificial neural networks, these two polarized views have been reflected by the rigidness of the *pre-destined* network connectivity and the adjustable connection weights. One speculation is that, if the observation that connectivity itself is flexible as much as the synaptic strength is gaining more ground in the field of biological neural network, numerical *vinaigrette* could be devised. The *Optimal Brain Growth* is, at least in the

very conceptual level, one such candidate model.

REFERENCES

[1] Alvarez-Sanchez, J. R., "Injecting knowledge into the solution of the two-spiral problem," *Neural Computing and Applications*, vol. 8, no. 3, pp. 265–272, 1999.

[2] Anastassiou, C. A., Perin, R., Markram, H., and Koch, C., "Ephaptic coupling of cortical neurons," *Nature Neuroscience*, 2011.

[3] Ash, T., "Dynamic node creation in backpropagation networks," in *International Joint Conference on Neural Networks*, vol. 2, IEEE, 1989.

[4] Belew, R. K., McInerney, J., and Schraudolph, N. N., "Evolving networks: Using the genetic algorithm with connectionist learning," *Artificial life II*, pp. 511–547, 1992.

[5] Benardos, P. G. and Vosniakos, G. C., "Optimizing feedforward artificial neural network architecture," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 3, pp. 365–382, 2007.

[6] Bernstein, A. V., Kuleshov, A. P., Sviridenko Yu, N., and Vyshinsky, V. V., "Fast aerodynamic model for design technology," in *Proceedings of West-East High Speed Flow Field Conference*, (Moscow), 2007.

[7] Bishop, C., "Exact calculation of the hessian matrix for the multilayer perceptron," *Neural computation*, vol. 4, no. 4, pp. 494–501, 1992.

[8] Bishop, C. M., *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[9] Bishop, C. M., *Pattern recognition and machine learning*. Springer New York:, 2006.

[10] Boahen, K., "Neuromorphic microchips," *Scientific American*, vol. 292, no. 5, pp. 56–63, 2005.

[11] Boozarjomehry, R. B. and Svrcek, W. Y., "Automatic design of neural network structures," *Computers and Chemical Engineering*, vol. 25, no. 7, pp. 1075–1088, 2001.

[12] Bull, L., "On the baldwin effect," *Artificial Life*, vol. 5, no. 3, pp. 241–246, 1999.

[13] Castillo, P. A., Merelo, J. J., Arenas, M. G., and Romero, G., "Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters," *Information Sciences*, vol. 177, no. 14, pp. 2884–2905, 2007.

[14] Choi, T. Y. W., Merolla, P. A., Arthur, J. V., Boahen, K. A., and Shi, B. E., "Neuromorphic implementation of orientation hypercolumns," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 6, pp. 1049–1060, 2005.

[15] Churchland, P. S. and Sejnowski, T. J., *The computational brain*. The MIT press, 1992.

[16] Cline, H., "Sperry and hebb: oil and vinegar?," *Trends in Neurosciences*, vol. 26, no. 12, pp. 655–661, 2003.

[17] Crick, F., *The astonishing hypothesis: The scientific search for the soul*. Scribner, 1995.

[18] Croegaert, M., Shapiro, S., Callahan, B., and Roach, J., "Evaluating intelligent fluid automation systems using a fluid network simulation environment," 2003. 13th Annual International Ship Control Systems Symposium, Orlando, FL.

[19] Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[20] da Silva, A. L. F., Cavalcanti, J. M. T., and Catalano, F. M., "Prediction of pressure distribution on wings using neural networks," 2008.

[21] dos Santos, C. and de Mattos, B. S., "Aerodynamic coefficient prediction of airfoils using neural networks," AIAA, 2008. 46 th AIAA Aerospace Sciences Meeting and Exhibit.

[22] Duvigneau, R. and Visonneau, M., "Hybrid genetic algorithms and neural networks for fast cfd-based design," in *9th AIAA/ISSMO Symposium and Exhibit on Multidisciplinary Analysis and Optimization*, vol. 4, (Atlanta), 2002.

[23] Erdi, P., *Complexity explained*. Springer Verlag, 2008.

[24] Fahlman, S. E. and Lebiere, C., "The cascade-correlation learning architecture," *Advances in neural information processing systems*, vol. 2, no. 2, pp. 524–532, 1990.

[25] Faller, W. E. and Schreck, S. J., "Real-time prediction of unsteady aerodynamics: Application foraircraft control and manoeuvrability enhancement," *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1461–1468, 1995.

[26] Faller, W. E. and Schreck, S. J., "Neural networks: applications and opportunities in aeronautics," *Progress in Aerospace Sciences*, vol. 32, no. 5, pp. 433–456, 1996.

[27] Floreano, D., Drr, P., and Mattiussi, C., "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[28] Floreano, D. and Urzelai, J., "Neural morphogenesis, synaptic plasticity, and evolution," *Theory in Biosciences*, vol. 120, no. 3, pp. 225–240, 2001.

[29] Friedman, J. H. and Stuetzle, W., "Projection pursuit regression," *Journal of the American statistical Association*, pp. 817–823, 1981.

[30] Fujii, K. and Dulikravich, G. S., "Parametric airfoils and wings," *Notes on Numerical Fluid Mechanics*, 1998.

[31] Ganguli, R., Chopra, I., Haas, D. J., and Engineer, A., "Detection of helicopter rotor system simulated faults using neural networks," *Journal of the American Helicopter Society*, vol. 42, p. 161, 1997.

[32] Gruau, F. and Whitley, D., "Adding learning to the cellular development of neural networks: Evolution and the baldwin effect," *Evolutionary Computation*, vol. 1, no. 3, pp. 213–233, 1993.

[33] Hacioglu, A., "Augmented genetic algorithm with neural network and implementation to the inverse airfoil design," 2004.

[34] Hagan, M. T. and Menhaj, M. B., "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[35] Happel, B. L. M. and Murre, J. M. J., "Design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, no. 6, pp. 985–1004, 1994.

[36] Hassibi, B. and Stork, D. G., "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, pp. 164–164, 1993.

[37] Haykin, S., *Neural networks: a comprehensive foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1994.

[38] Hazarika, N., Tuncer, H., and Lowe, D., "An inverse design procedure for airfoils using artificial neural networks," ICAS, 1998.

[39] Hendrickson, R. and Rajkovic, D., "5.1 the importance of drag."

[40] Hernandez-Espinosa, C. and Fernandez-Redondo, M., "Multilayer feedforward weight initialization," vol. 1, pp. 166–170 vol. 1, IEEE, 2001. Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on.

[41] Hinton, G., "The next generation of neural networks," 2007.

[42] Hortsch, M. and Umemori, H., *The sticky synapse*. Springer.

[43] Hsken, M., Jin, Y., and Sendhoff, B., "Structure optimization of neural networks for evolutionary design optimization," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 1, pp. 21–28, 2005.

[44] Hwang, J. N., You, S. S., Lay, S. R., and Jou, I. C., "The cascade-correlation learning: A projection pursuit learning perspective," *Neural Networks, IEEE Transactions on*, vol. 7, no. 2, pp. 278–289, 1996.

[45] Jabr, F., "Ibm unveils microchip based on the human brain," *The New Scientist*, vol. 211, no. 2827, pp. 20–20.

[46] Jeong, S., Chiba, K., and Obayashi, S., "Data mining for aerodynamic design space," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 11, pp. 452–469, 2005.

[47] Jia, J. and Chua, H. C., "Solving two-spiral problem through input data representation," in *IEEE International Conference on Neural Networks*, vol. 1, pp. 132–135 vol. 1, IEEE, 1995.

[48] Kitano, H., "Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms," *Physica D Nonlinear Phenomena*, vol. 75, pp. 225–238, 1994.

[49] KrishnaKumar, K., "Optimization of the neural net connectivity pattern using a backpropagation algorithm," *Neurocomputing*, vol. 5, no. 6, pp. 273–286, 1993.

[50] Kurkov, V., "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*, vol. 5, no. 3, pp. 501–506, 1992.

[51] Kwok, T. Y. and Yeung, D. Y., "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *Neural Networks, IEEE Transactions on*, vol. 8, no. 3, pp. 630–645, 1997.

[52] Le Cun, Y., Denker, J. S., Solla, S. A., Howard, R. E., and Jackel, L. D., "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, no. 1, p. 1990, 1990.

[53] Levin, E., Tishby, N., and Solla, S. A., "A statistical approach to learning and generalization in layeredneural networks," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1568–1574, 1990.

[54] Li, G., Alnuweiri, H., Wu, Y., and Li, H., "Acceleration of back propagation through initial weight pre-training with delta rule," pp. 580–585 vol. 1, IEEE, 1993. IEEE International Conference on Neural Networks.

[55] MacKay, D. J. C., "Bayesian interpolation," *Neural computation*, vol. 4, no. 3, pp. 415–447, 1992.

[56] Markram, H., "The blue brain project," *Nature Reviews Neuroscience*, vol. 7, no. 2, pp. 153–159, 2006.

[57] Mason, W. H., "Configuration aerodynamics," 2006.

[58] McCulloch, WS, P. W., "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, 1943.

[59] Moon, K., *Self-Reconfigurable Ship Fluid-Network Modeling for Simulation-Based Design*. PhD thesis, Georgia Institute of Technology, 2010.

[60] Nguyen, D. and Widrow, B., "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," pp. 21–26 vol. 3, IEEE, 1990. Neural Networks, 1990., 1990 IJCNN International Joint Conference on.

[61] Noble, W. S., "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[62] Nolfi, S., Parisi, D., and Elman, J. L., "Learning and evolution in neural networks," *Adaptive Behavior*, vol. 3, no. 1, p. 5, 1994.

[63] Noyes, J. L., "Neural network training," 1997.

[64] Oyama, A., Obayashi, S., and Nakamura, T., "Real-coded adaptive range genetic algorithm applied to transonic wing optimization," *Applied Soft Computing*, vol. 1, no. 3, pp. 179–187, 2001.

[65] Pierret, S. and Van den Braembussche, R. A., "Turbomachinery blade design using a navier-stokes solver and artificial neural network," *ASME Journal of Turbomachinery*, vol. 121, no. 3, pp. 326–332, 1999.

[66] Rai, M. M., "Towards a hybrid aerodynamic design procedure based on neural networks and evolutionary methods," *AIAA paper*, vol. 3143, 2002.

[67] Rai, M. M., "Robust optimal aerodynamic design using evolutionary methods and neural networks," *AIAA paper*, vol. 778, 2004.

[68] Rai, M. M. and Madavan, N. K., "Aerodynamic design using neural networks," *AIAA journal*, vol. 38, no. 1, pp. 173–188, 2000.

[69] Ravindra, K., Shende, N. V., and Balakrishnan, N., "Performance of code hifun in spices 09," in *11th AeSI Annual CFD symposium*, (Indian Institute of Science, Bengaluru), pp. 11–12, 2009.

[70] Reed, R., "Pruning algorithms-a survey," *Neural Networks, IEEE Transactions on*, vol. 4, no. 5, pp. 740–747, 1993.

[71] Rocha, M., Cortez, P., and Neves, J., "Simultaneous evolution of neural network topologies and weights for classification and regression," *Lecture notes in computer science*, vol. 2902, p. 5966, 2005.

[72] Rocha, M., Cortez, P., and Neves, J., "Evolution of neural networks for classification and regression," *Neurocomputing*, vol. 70, no. 16-18, pp. 2809–2816, 2007.

[73] Rojas, R. and Feldman, J., *Neural networks: a systematic introduction*. Springer, 1996.

[74] Rumelhart, D. E., Hintont, G. E., and Williams, R. J., "Learning representations by back-propagating errors," *NATURE*, vol. 323, p. 9, 1986.

[75] Sanderson, C., "Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments," tech. rep., Technical report, NICTA.

[76] Scott, R. C., "Active control of wind-tunnel model aeroelastic response using neural networks," vol. 3991, p. 232, 2000. Proceedings of SPIE.

[77] Sexton, R. S., McMurtrey, S., and Cleavenger, D., "Knowledge discovery using a neural network simultaneous optimization algorithm on a real world classification problem," *European Journal of Operational Research*, vol. 168, no. 3, pp. 1009–1018, 2006.

[78] Shahrokhi, A. and Jahangirian, A., "Airfoil shape parameterization for optimum navier-stokes design with genetic algorithm," *Aerospace Science and Technology*, vol. 11, no. 6, pp. 443–450, 2007.

[79] Stahara, S. S., *Operational manual for two-dimensional transonic code TSFOIL*, vol. 3064. NASA, 1978.

[80] Stanley, K. O. and Miikkulainen, R., "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[81] Steering Committee for the Decadal Survey of Civil Aeronautics, N. R. C., *Decadal Survey of Civil Aeronautics: Foundation for the Future*. Washington D.C.: The National Academic Press, 2006.

[82] Su, W., Gao, Z., and Zuo, Y., "Application of rbf neural network ensemble to aerodynamic optimization," 2008.

[83] Sutherland, C. T., "Online genetic re-training of a neural network control systems for wind tunnels," 2000.

[84] Suykens, J. A. K. and Vandewalle, J., "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[85] Takenaka, K., Obayashi, S., Nakahashi, K., and Matsushima, K., "The application of mdo technologies to the design of a high performance small jet aircraft-lessons learned and some practical concerns," *AIAA paper*, vol. 4797, 2005.

[86] Turney, P., Whitley, D., and Anderson, R. W., "Evolution, learning, and instinct: 100 years of the baldwin effect," *Evolutionary Computation*, vol. 4, no. 3, 1996.

[87] Vavalle, A. and Qin, N., "Iterative response surface based optimization scheme for transonic airfoil design," *Journal of Aircraft*, vol. 44, no. 2, p. 365, 2007.

[88] Wagner, G. P., Pavlicev, M., and Cheverud, J. M., "The road to modularity," *Nature Reviews Genetics*, vol. 8, no. 12, pp. 921–931, 2007.

[89] Whitley, D., Gordon, S., and Mathias, K., "Lamarkian evolution, the baldwin effect, and function optimisation," in *International Conference on Evolutionary Computation, The Third Conference on Parallel Problem Solving from Nature*, (Jerusalem, Israel), p. 6, Springer, 1994.

[90] Wilamowski, B. M., "Neural network architectures and learning algorithms," *Industrial Electronics Magazine, IEEE*, vol. 3, no. 4, pp. 56–63, 2009.

[91] Wilamowski, B. M., Hunter, D., and Malinowski, A., "Solving parity-n problems with feedforward neural networks," vol. 4, pp. 2546–2551 vol. 4, IEEE, 2003. Proceedings of the International Joint Conference on Neural Networks.

[92] Wilamowski, B. M. and Yu, H., "Neural network learning without backpropagation," *Neural Networks, IEEE Transactions on*, vol. 21, no. 11, pp. 1793–1803.

[93] Yao, X., "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[94] Yao, X. and Islam, M. M., "Evolving artificial neural network ensembles," *IEEE Computational Intelligence Magazine*, vol. 3, no. 1, pp. 31–42, 2008.

[95] Zedda, M. and Singh, R., "Fault diagnosis of a turbofan engine using neural networks-a quantitative approach," 1998.